Applied Soft Computing 55 (2017) 462–479

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Ant colony optimization based hierarchical multi-label classification algorithm

Salabat Khan [a,*], Abdul Rauf Baig [b]

[a] Dept. of Computer Science, Comsats Institute of Information Technology, Attock, Pakistan
[b] College of Computer and Information Sciences, Al Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia

ARTICLE INFO

Article history:
Received 17 July 2014
Received in revised form 10 February 2017
Accepted 21 February 2017
Available online 24 February 2017

Keywords:
Hierarchical multi-label classification
Ant colony optimization
Hierarchical single label classification
Bioinformatics data sets with gene ontology and FunCat
Protein function prediction
Correlation based IF-THEN rule list
HmAntMiner-C

ABSTRACT

There exist numerous state of the art classification algorithms that are designed to handle the data with nominal or binary class labels, where a sample belongs to only a single class label. In these problems, known as flat classification problems, class labels are independent of each other. Unfortunately, on the other hand, less attention is given to the genre of classification problems where samples may belong to several classes and at the same time the class labels are organized based on a structured hierarchy; such as gene ontology, protein function prediction, test scores, web page categorization, text categorization etc. This article presents a novel Ant Colony Optimization based hierarchical multi-label classification algorithm that can handle such a complex instance of classification problems and can incorporates the given class hierarchy during its learning phase. The algorithm produces IF-THEN ordered rule list to learn a comprehensible model which can easily be verified by experts. It exploits positive correlation between the domain values of two related attributes to improve the discrimination power of resultant classification model, up to a significant level. The paper contains rich details regarding hierarchical single label (or single path) and multi-label classification problems and different categories of corresponding solutions. The proposed method is evaluated on sixteen most challenging bioinformatics datasets; some of these containing hundreds of attributes and thousands of class labels. At the end, the proposed method is compared with four recent state of the art hierarchical multi-label classification algorithms. The empirical evaluation confirms the promising ability of the proposed technique for hierarchical multi-label classification task.

1. Introduction

Data mining techniques can be used to extract implicit, previously unknown and potentially useful [1] patterns and knowledge of interests from vast data stores for varied purposes. It is a highly interdisciplinary field that has brought together researchers from multiple fields e.g., statisticians, mathematicians, computer scientists, biologists, data visualization experts, working on state of the art novel computational and statistical techniques in order to understand complex physical and biological systems. These techniques can help the companies in gaining competitive advantageous through effective decision making process based on their current available data. In order to support the decision making process, classification is considered, one of the most effective techniques under the domain of Data Mining. Classification task is successfully used in a myriad of applications e.g., decision making, fraud detection, medical diagnosis, credit scoring, customer relationship management, character recognition, speech recognition, protein function prediction etc [1–9,11]. A classification problem encompasses two main stages; in the first stage, a collection of training samples from the input data is used to train the classification model (classically known as hypothesis) and in the second stage the model is tested over the samples previously unseen by the model in order to analyze its generalization capability.

There are a number of classification algorithms that are proposed in past, such as decision trees, neural networks, k-nearest neighbor classifiers, logistic regression and support vector machines (SVM) etc. [2–4]. The techniques such as neural networks and support vector machine etc. generate an incomprehensible classification model that is obscure to human, while other techniques (decision tree e.g.,) output classification models that are considered comprehensible (e.g., IF-THEN rule list of decision trees) to the experts working in different domains. The comprehensible classification models can easily be validated by the end users based

* Corresponding author.
E-mail address: salabat.khan@nu.edu.pk (S. Khan).

http://dx.doi.org/10.1016/j.asoc.2017.02.021
1568-4946/

on their specialized knowledge. These classical flat classification techniques resulted in good performance when evaluated in the problems related to various domains and considered computationally efficient (e.g., SVM), robust to noisy data (e.g., decision trees) and easy to learn (e.g., k-nearest neighbors). However, most of these classification techniques are designed to handle the data with binary or nominal class labels that are independent of each other (meaning that class labels are flat). The above-mentioned classification algorithms are unable to handle the classification problems where the class labels are dependent and are organized based on a class hierarchical structure (CHS). These classification problems are known as hierarchical classification [5] and discussed in follows.

For hierarchical classification, the class labels that are to be predicted are naturally organized as a class hierarchy (also known as CHS). The CHS is typically represented as a tree or Directed Acyclic Graph (DAG), see e.g., Fig. 1(a, b). The class labels in the hierarchy are represented as nodes and the relationships between the class labels are shown with undirected edges. The difference between tree and DAG representation is simple that in the tree hierarchical structure, a node (representing a class label) can have only one parent whereas no such restriction is imposed in DAG hierarchical structure. Thus, based on the Gene Ontology (a CHS for protein function prediction used in the experiments), a node in the hierarchy can have more than one direct ancestors. The class hierarchy defines the parent-child (IS-A) relationship between class labels at different levels of the hierarchy. Predicting a single class label in the hierarchy implies that all the ancestor class labels are also predicted, in other words, a single class label is a path from root node to the predicted child node (explained later) satisfying the IS-A relationship. During learning a classification model for a hierarchical classification problem, this class hierarchy must be incorporated for gaining better performance which pose a more complex scenario than flat single label classification problem.

In real world, hierarchical classification has applications in various domains. This research work mainly focuses on hierarchical protein function prediction where the CHS is defined under the schemes of MIP's FunCat and Gene Ontology (GO), specifically "functional genomics" domain. Based on the FunCat scheme [38], the class hierarchical structure is represented as a tree. Whereas on the other hand, GO structure [6] represents the relationship among the protein functions using directed acyclic graph (DAG) and therefore considered a challenging instance of hierarchical classification problems as compared to the problems using FunCat scheme. There is a large amount of uncharacterized protein data available for analysis that has lead to an increased interest in computational methods to support the investigation of the role of proteins in an organism [7,8]. Protein classification schemes (such as the Gene Ontology [6]) are organized in a hierarchical structure, allowing the annotation of protein at different levels of detail. It is well known that a protein can perform more than one function and the class labels are organized based on a hierarchy (FunCat and GO, in our case), thus makes the problem a hierarchical multi-label classification problem. Analyzing the functions of proteins in different organisms is crucial to improve biological knowledge, diagnosis and treatment of diseases. It is not possible to conduct the biological experiments for the functional essay analysis of every uncharacterized protein due to involvement of high cost and human based analysis which is cumbersome, as the task is so complex to be carried out manually [9]. It therefore raise the need of the development of computational methods (especially related to data mining domain) to be used for this purpose.

Based on class label(s) associated with an example, the hierarchical classification is further categorized in two different categories [9]:

1) **Hierarchical Single Label (path) Classification:** In this type of classification task, the class labels are organized in a hierarchy and at the same time, an example/instance have multiple class labels associated with it. However, an example is associated with only *one class label at any level* of the class hierarchy (remember that single class label represents a path in the hierarchical structure from root node to that single class node and therefore, all the ancestor class labels are also associated with the example).
2) **Hierarchical Multi-label (path) Classification:** In this type of classification task, the class labels are organized in a hierarchy and at the same time, an example/instance have multiple class labels associated with it. However, an example can be associated *with more than one class label at any level* of the hierarchy (multi-paths in the hierarchy).

Based on the type of condition placed on the depth (level) of the predicted class labels [9,10], hierarchical classification task has further two categories. First is the mandatory leaf-node prediction (MLNP) and second is the optional leaf-node prediction (OLNP) or non-mandatory leaf-node prediction (NMLNP). For mandatory leaf-node prediction problems, a classification model must predict the leaf class label(s) for a given test example. The optional leaf-node prediction problems are flexible in the sense that it is not mandatory for a classification model to predict the leaf class label(s) for a given test example. In this research work, a hierarchical classification algorithm is proposed based on ant colony optimization for hierarchical multi path classification, considering only the MLNP case. Considering the prediction requirement for the class labels, we can indeed say that our technique is supposed to decide about all the class labels, whether to be present or not, for a predicted IF-THEN rule.

Looking into the class hierarchy (e.g., Fig. 1), the nodes at the upper levels, represent more general class labels whereas the nodes at the lower levels, show the more specific class labels. A class label (concept to learn) is dependent on its direct ancestor(s) and this relationship recursively reaches to the root of the class hierarchy. Classes at the upper level of the hierarchy (i.e., general classes) are easy to predict as numerous examples/instances related to them are available (to the hypothesis learner) but classes at the deeper level of the hierarchy (i.e., specific classes) are difficult to predict as less information is available to discriminate among them and the data at these levels is highly skewed. The root of the hierarchy, represents the case where the class label is unknown since it does not contain any knowledge about the class label of an example, and therefore can be labeled as 'dummy/any'.

In order to clearly describe what do we exactly mean by class taxonomy, we follow the definition provided in the survey [10]. This definition is based on (C, Δ) where 'C' is a finite set containing all the class labels of the hierarchy and Δ correspond to 'IS-A' relationship. According to this definition, 'IS-A' relationship between nodes in the class hierarchical structure (CHS) is asymmetric, anti-reflexive and transitive: (see e.g., Animal Hierarchy in Fig. 2)

1) The only one greatest element in the hierarchy is represented by root node
2) $\forall li, lj \in C$, if $li \Delta lj$ then $lj\ li$ {e.g., considering the Animal hierarchy if li is a Dog and lj is a Doberman then Doberman 'IS-A' dog but not all dogs are Doberman}
3) $\forall li \in C$, $li \Delta li$
4) $\forall li, lj$ and $lk \in C$, if $li \Delta lj$ and $lj \Delta lk$ then $li \Delta lk$ (See for instance, the Animal hierarchy)

Most general class label in the CHS is placed on the top (e.g., Animal in Fig. 2) and the one which is present in all the data samples, thus gives no knowledge to the training phase of the classifier. Dog and Cat are two class labels that are more specific than their parent

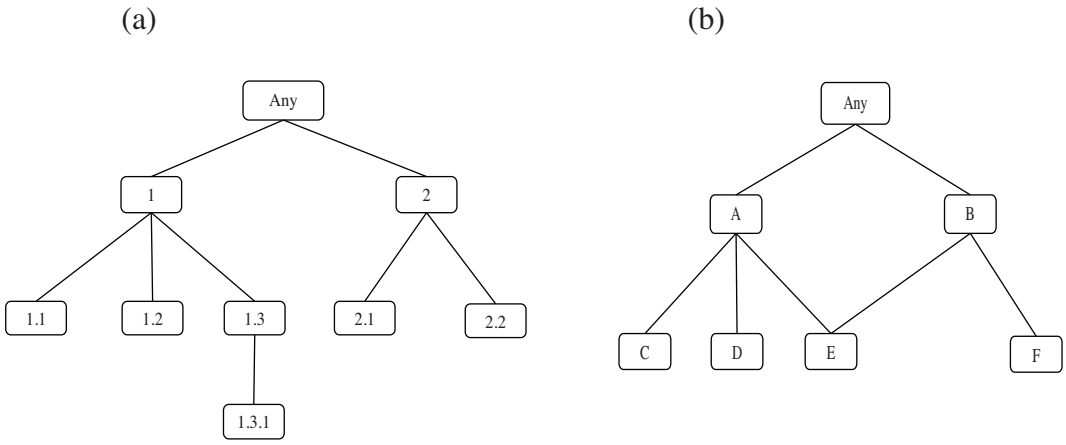

Fig. 1. a) Tree Hierarchy b) DAG Hierarchy (Two parents for 'E').

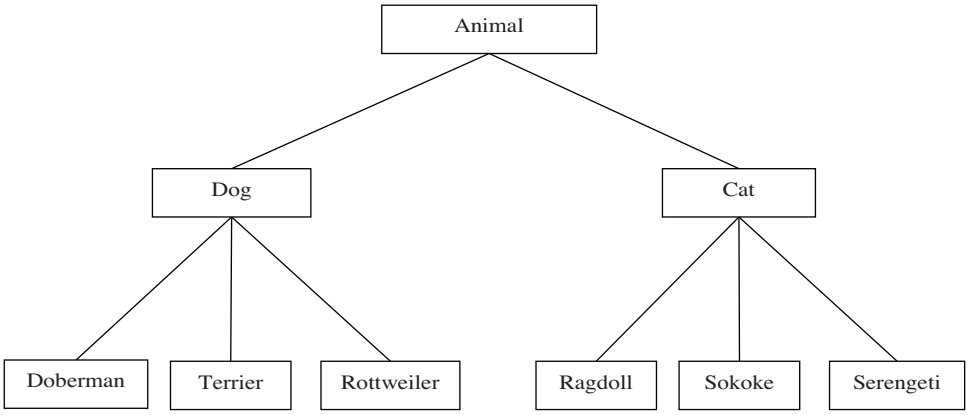

Fig. 2. Partial Animal Hierarchy.

'Animal'. Class labels present at the third level are most specific class labels and it is challenging to correctly predict these class labels; as number of data instances per class label are very small in size to help the classification learner perform discrimination. There is always a tradeoff between generality and specificity in hierarchical classification. The general class labels are easy to predict and in turn the classification error rate will be low but this gives less information about the concept, we are learning. On the other side, predicting specific class labels offer more information about the concept, we are learning, but the classification model will be more prone to prediction error.

The remainder of this paper is organized as follows. In the next section, we review related research work and different categories of solutions for hierarchical classification task. In Section 3, we present the basics and the background of ant colony optimization meta-heuristic. In Section 4, the proposed method is discussed. Subsequently, in Section 5, we present some simulation results to show the promising ability of our technique. Finally, Section 6 will conclude this work.

2. Related work

In this section, the focus is on the discussion about different categories of solutions to handle the hierarchical classification task where most of the discussion is based on the guidelines presented in [10]. Multiple criterions can be considered to categorize the hierarchical classification methods. The first criterion is about the type of class hierarchical structure (e.g., Tree or DAG) that is handled by the algorithm. The second criterion is related to the depth of class hierarchy being targeted during the classification procedure e.g., MLNP, OLNP etc. The third criterion is about how the class hierarchical structure is incorporated and explored. Considering the current research literature about hierarchical classification, most of the strategies are based on a top-down (or local) classification system where a set of local classifiers are exploited to handle the underlying hierarchical classification problem (e.g., using a single classifier per node, per parent node or per level in the class hierarchy). The other solution strategy to deal with hierarchical classification problem is known as big-bang (or global) classification system, when a single classifier handling the entire class hierarchy (at once) is used, looking at the problem instance from a global perspective. The focus of article is on global hierarchical classification models, the readers are kindly referred to [49] for a detailed discussion on local classifier approaches for hierarchical classification.

2.1. Big-bang or global approach

In this approach, all the related class labels as per the given CHS are considered at once from a global point of view. As a result, only a single global classification model is built which is compact in size and is typically considerably smaller as compared to the total model size of all the local models constructed using any of the local classifier approaches. This approach is known as the big-bang approach, also called "global" learning [10]. The global approach provides solution to both the structures, Tree & DAG. The class prediction inconsistencies as per the given CHS are avoided during the testing phase. It is however a complex task to model a global hierarchical classification technique. Moreover, the parallel learning of

multiple classifiers is now not useful as it was for local classifier scheme.

In order to support the above mentioned properties, some local and big-bang hierarchical classification algorithms are discussed as follows. In the work of Kiritchenko et al. [29], the class label(s) of samples are expanded to make it sure that all their ancestor(s) are also present in the samples. Later, a multi-label classification algorithm is used to train the classification model to make further predictions on test data. However, in some cases, a post-processing step is required to handle hierarchical inconsistencies in the class label space. Rousu et al. [39] presents a maximum margin Markov networks framework that is based on kernels to address hierarchical multi-label classification problem. This strategy does not require a post-processing step to handle the constraint of hierarchically consistent predictions. These strategies learn a classification model (in the form of complex mathematical function) which cannot easily be interpreted and verified by the users (e.g., radiologists, finance experts, biologists and physicians etc) and can be considered as 'black box' classification approaches. There exist strategies to convert the kernel based classification models into comprehensible IF-THEN rule list form; this usually however decreases the generalization performance [37] of the classifier. In follows, we discuss few comprehensible classification algorithms for hierarchical classification task.

In the research work of Yen-Liang Chen et al. [1], flat decision tree is extended to handle the data arranged with hierarchical class labels. The algorithm proposed in their work use an extended entropy measure. The selection of an attribute as a node in the decision tree is based on this modified evaluation measure known as Hierarchical-Entropy value [1]. The two main objectives, prediction accuracy and prediction specificity are optimized and in order to handle the tradeoff between these two objectives, a scoring mechanism is used that combines the values of these two objective measures to assign the predicted class labels for an example in the dataset. The algorithm however, cannot handle the DAG hierarchical structure and hierarchical multi-label classification problem. The distance between the real and predicted class labels is not taken into consideration due to which the performance evaluation measure is not very representative to the hierarchical classification task. The algorithm achieved higher percentage accuracy as compared to its ancestor i.e., flat decision tree C4.5 [2] with a compact decision tree in terms of number of nodes in the tree. Clare et al. [40] extended the base line C4.5 decision tree algorithm to handle the hierarchical classification task. In this new adapted version of decision tree, a leaf in the tree predicts a vector of boolean values for indicating the presence or absence of class labels.

Another worth mentioning, decision tree based method to handle the hierarchical multi-label classification task is proposed in the work of Celin Vens et al. [12] known as CLUS-HMC. Basically, three different strategies are investigated: 1) training a decision tree for each class label, separately, called *CLUS-SC*, 2) training a decision tree in top-down fashion and at the same time, considering hierarchically consistent (i.e., a class is associated with a sample only if any of it(s) parent(s) belong to the sample, already) predictions only, called *CLUS-HSC* and 3) training a single global classification model predicting all the class labels at once, called *CLUS-HMC*. These three strategies are well-known to the research community due to their efficient handling of hierarchical classification problems and are thus chosen as competitors for evaluation of proposed classifier.

Holden et al. [41] proposed a hybrid method based on Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) in order to improve the performance of top-down hierarchical classification approach. Specifically, several candidate classifiers are trained per a single class node and the selection of best among them is done through the hybrid optimization strategy. This work is originally an extension to an earlier work [42] where the selection of best is done through greedy strategy. Later, they have further improved this strategy by using the collective decisions of a committee of classifiers rather than using a single best classifier for prediction at a particular level of CHS [43]. PSO is used to optimize the weights for the decision of each classifier. Both of these approaches are unable to handle DAG hierarchical structure and hierarchical multi-label classification problem.

The hAntMiner (stands for hierarchical AntMiner) is the first ACO based algorithm proposed by Fernando et al. [7] to solve hierarchical single-label classification task as an extension to the flat classification algorithm called AntMiner [14]. The algorithm is tested on protein functions prediction data sets defined under the structure of Gene Ontology (GO). The procedure to dynamically handle the continuous attributes is borrowed from the earlier work of the same author [15].hAntMiner cannot handle hierarchical multi-label classification task and is therefore not investigated in our experiments.

In order to address the limitations of hAntMiner [7], Fernando et al. [8] presented an extended version, called hmAntMiner which can also handle the hierarchical multi-label classification task. hmAntMiner is another ideal competitor to the proposed algorithm. In fact, our algorithm which is an extension to an earlier work [28], has few parts in common to that of hmAntMiner; as all of the current versions of ant-based classification model are originally inspired from the work of Parpinelli et al. known as AntMiner [14], thus share a number of common traits. The salient differences between both the algorithms are mentioned here briefly, more insight discussions are given in Section 4. The difference between the designs of these two algorithms highlight our contributions.

- The search space topology of both the algorithms (hmAntMiner and hmAntMiner-C) is different. hmAntMiner use Mesh topology in order to connect the vertices (representing attribute-value pair) in the search graph. hmAntMiner-C on the other hand, use the layering of attribute-value pair, as a grid and directed acyclic graph (DAG) topology (Fig. 3) which makes the presentation of search space simpler than its competitor. The constraints on the moves of ants can now be easily implemented and become more comprehensible to the users as discussed in Section 4.
- The proposed algorithm need discretization as a pre-processing step in order to handle the continuous valued attributes due to which it is easily adopted for the correlation based heuristic function (Section 4.7). hmAntMiner can handle nominal as well as continuous attributes.
- hmAntMiner-C reduces the search space complexity dynamically by removing those components from the search space that are no more present in the current training data. This makes the algorithm computationally less demanding as compared to hmAntMiner.
- Our algorithm use a more accurate correlation based heuristic function based on the distances in the CHS space which is an extended version of the heuristic used in hmAntMiner and CLUS-HMC. We use the current history of the ant tour in order to guide the ant for the next move and therefore the selection of vertices in the search space become accurate and appropriate to the context at that moment.
- The representation of pheromone matrix and evaporation process is different, see for detail, Section 4.

3. Basic ACO algorithm and its applications

Ant colony optimization (ACO) is a famous meta-heuristic proposed by M. Dorigo [19–21] in the early 90's. Based on its learning style, it is considered a suitable strategy to solve complex combinatorial optimization problems. ACO is a branch of swarm intelligence

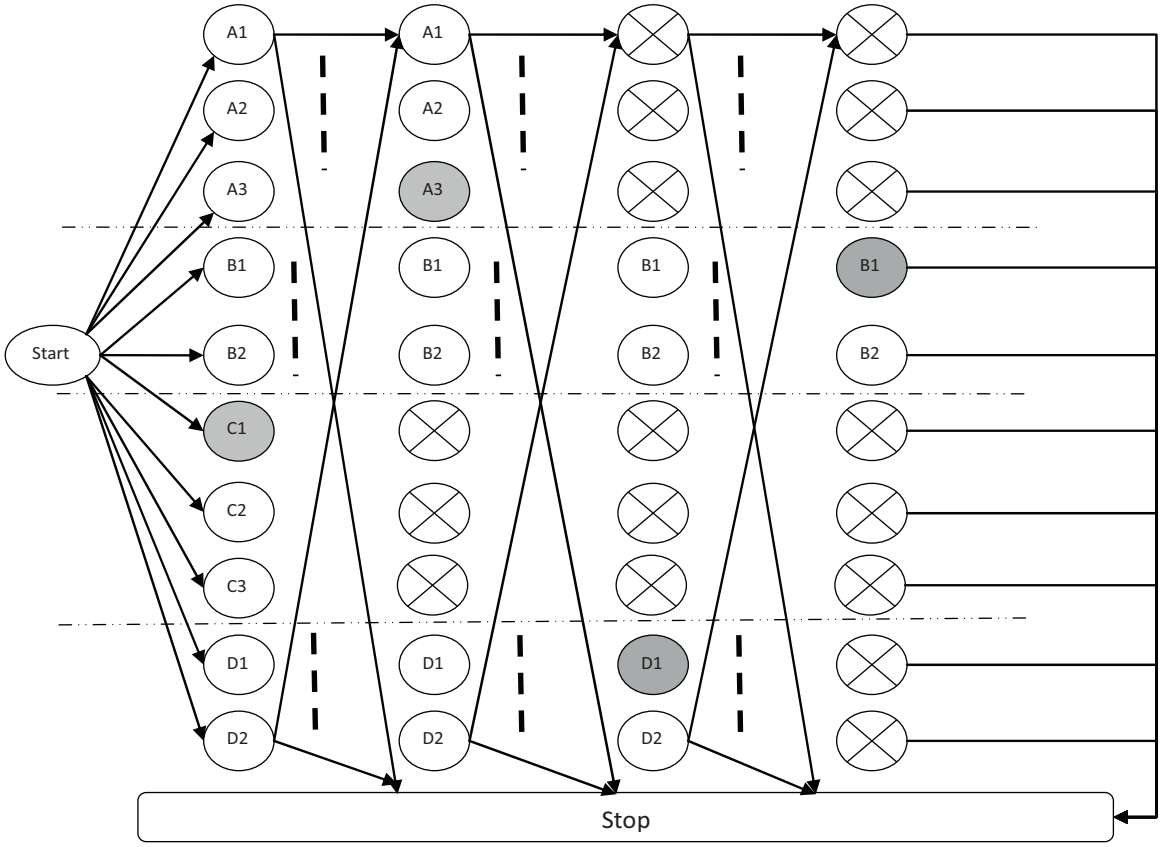

Fig. 3. An example search space of hmAntMiner-C for constructing rule antecedent.

[16–18] (which deals with the collective behavior of small and simple entities) & has been used in many real world applications e.g., Travelling Salesperson Problem (TSP), quadratic assignment, vehicle routing, connection oriented and connectionless network routing, sequential ordering, graph coloring, shortest common super sequence, single machine tardiness, multiple knapsack etc. Recently, it has been successfully applied to the problems related to data mining [23–25,48].

Ant colony optimization is inspired after the collective behavior of biological ants during the search for food sources. It is observed that ants propagate their findings about the trails in the search space by influencing the environment. For this purpose, ants spread a chemical substance called pheromone in the search environment. The ants communicate with each other by means of pheromone. The coordination between the ants in the colony is not direct and this indirect communication through modifying the environment is known as stigmergy. Although, each individual ant as an entity/agent has only limited capabilities, the complete swarm exhibits complex overall behavior [22]. The ants that arrive in the vicinity are more likely to take the path with higher concentration of pheromone than the paths with lower concentrations. In other words, the desirability of ants for selecting the paths is proportional to the pheromone concentrations. The pheromone evaporates with time and if new pheromone is not added, the older one dwindles away [27,28].

In case, more than one e.g., two paths between a food source and the ant nest are initially discovered by some ants, subsequent ants will soon switch over the shorter paths. The reason is that the ants shuttle tours over shorter paths take less time than the tours over longer paths, and hence the rapid rate of pheromone concentration over the shorter paths divert the swarm from longer paths toward the shorter paths. The ants can find the shorter path but what if an obstacle is placed in an already established path? In case, if an established path is blocked, some ants will first go to the left and some to the right of the obstacle with equal probability. However, an ant taking the shorter of the two paths will return earlier than the ant taking the longer path and hence the pheromone on the shorter path will be enhanced and subsequent ants will have a higher probability of taking the shorter path. Soon a new shorter path which bypasses the blockage will be established. The more ants follow a given trail, the more attractive that trail becomes and is followed by other ants [28].

ACO meta-heuristic usually models the real world environment of ants in the form of a graph. The vertices of the graph represent the components of a candidate solution. The edges are traversed by ants to create the trails. The artificial pheromone values are associated with the edges and updated based on the quality of a trail. The more the quality of a trail the more the concentration of pheromone is carried out and the more the trail become attractive for the ants. An artificial ant constructs a candidate solution to the problem by adding solution components one by one. Before the construction of complete candidate solution, usually a problem dependent heuristic is used in collaboration with pheromone values to guide the ants moves. Subsequently, as time passed, ants construct their solutions one by one and guide each other to find better and more better solutions. The components with higher pheromone concentrations are thus identified as contributing to a good solution and repeatedly appear in the solutions. Usually, after sufficient iterations, the ants converge towards a good, if not the optimal, solution.

For the application of ACO to a problem we have the following requirements [28]:

- The ability to represent a complete solution as a combination of different components.

– There should be a method to determine the fitness or quality of the solution.
– A heuristic measure for the solution's components (this is desirable but not necessary).

Since its inception, ACO has been applied to solve many real world optimization problems [17–19]. It is by design suited to discrete optimization problems, such as, quadratic assignment, job scheduling, subset problems, network routing, vehicle routing, load dispatch in power systems, bioinformatics, and, of course, data mining which is the subject of this paper.

4. Method

In this section, we discuss our method for hierarchical multi-label classification task. The algorithm is an extended version of work [28] and exploits the distance based correlation between the domain values of two attributes (discussed later) to find most accurate & representative IF-THEN rule list for the input data. The distance or variance in the class label space is used to guide the swarm in the right direction. The usage of current context in terms of ants tours, improve the performance of the algorithm up to a significant level. We begin with the definition of the problem tackled in this work followed by a brief general description of the proposed algorithm. Afterwards, each and every stage of the approach is further discussed in a fair amount of details for concrete understanding. The stages are: search space formation, rule construction based on pheromone and a variance based correlation heuristic function, rule evaluation, rule pruning, pheromone update and evaporation, stopping criteria etc.

4.1. Problem definition

Given an input data set and a class hierarchical structure (CHS) e.g., tree/DAG, the goal is to build a list of IF-THEN rules, such that two objectives i.e., prediction accuracy and prediction specificity is maximized or specifically, the area under precision recall curve (discussed later) is maximized. The class hierarchical structure (CHS) contains a root class label that covers all the examples in the given input data set e.g., as shown in Fig. 2. Based on the given CHS, samples in the dataset can belong to multiple class labels at a given level in the CHS; thus presenting a hierarchical multi-label classification problem.

4.2. General description

The core of the algorithm is the incremental construction of a classification rule of the type:

$$\begin{aligned}&\textbf{IF} < \text{term}_1 \ \textbf{AND}\ \text{term}_2 \ \textbf{AND}\ldots > \\ &\textbf{THEN} \\ &< \text{class}_1\,[\text{probability}],\ \text{class}_2\,[\text{probability}], \ldots\ldots,\ \text{class}_k\,[\text{probability}] >\end{aligned} \tag{4}$$

A term in the antecedent part is an attribute-value pair. A number of relational operators may be used to relate the attribute with one of its possible domain values to create a term. In this work, we use "=" as the only possible relational operator. An example term is "Color = yellow". The attribute's name is "color" and "yellow" is one of its domain values. Since we use only "=" operator, any continuous (real-valued) attributes present in the data have to be discretized in a preprocessing step. The consequent of the rule contain the class labels that are hierarchically related based on the given CHS. All the class labels are present in the consequent along with their probability values. The probability value of a class label shows its degree of association with the rule, based on the conditions present in the antecedent part. During classification of unseen data, a threshold value is selected by user based on which the class labels are assigned to the unseen samples. Specifically, the class labels that have probability value greater than or equal to the given threshold, are assigned to the given test sample. In order to calculate class probability values for the consequent part of a rule, it is sensible to use the examples covered by the rule itself. Getting inspired from hmAntMiner [8], the probability of a class label '*i*' (in the given CHS) for a given rule '*r*' is calculated as follows:

$$probability_{r,i} = \frac{|Cov_r, Class\_Label_i|}{|Cov_r|} \tag{5}$$

Where $|Cov_r, Class\_Label_i|$ represents the number of training examples covered by rule '*r*' and contain the class label '*i*' in their class attribute, which is divided by $|Cov_r|$ (total number of training examples covered by rule '*r*'). In this way, two important requirements of hierarchical multi-label classification task are fulfilled. First, the algorithm can predict unrelated class labels for a given a test example or in other words, multiple class labels at any of the levels in CHS can be predicted, simultaneously. Second requirement is related to hierarchical consistent class prediction i.e., it is not possible to predict a class label until one of its parent is already predicted (as parent class label probabilities will always be greater than or equal to their child class label probabilities). The stages constituting the proposed approach are further discussed in the following subsections.

4.3. Search space and tour construction

The solution to a particular problem based on Ant Colony Optimization algorithm starts with designing a problem search space in which the ants conduct the search to find the candidate solutions. The search space for hmAntMiner-C is defined with the help of input dataset, represented as DAG topology given in Fig. 3. The overall search space is described with the layering of attribute-value pairs (terms) based on grid like topology. The dimensions (or coordinates) of the antecedent construction graph (search space) are the attributes of the dataset (excluding the class attribute). The possible values of an attribute constitute the range of values for the corresponding dimension in the search space. For example, a dimension called 'Weather' may have three possible values {sunny, windy, rainy}. The ant visits a dimension and choose one of its possible values to form an antecedent condition of the rule (e.g., Weather = sunny). The total number of terms for a dataset is given:

$$Total\_terms = \sum_{n=1}^{a} b_n \tag{6}$$

Where '*a*' is the total number of attributes (excluding the class attribute) and 'b_n' is the number of possible values in the domain of attribute 'A_n'. When a dimension has been visited, it cannot be visited again by the same ant, as the conditions of the type "Color = Red **OR** Color = Green" are not allowed. The antecedent sub-graph is flexible in the sense that an ant can pick a term from any of its dimensions and there is no strict ordering in which the dimensions are required to be visited.

The ant starts it tour from the vertex labeled "Start" and keep moving in the search space from left to right (based on the information available in the form of pheromone and heuristic value) subject to some constrains. Only one node is allowed to be selected in a column of the search space. An example search space is shown in Fig. 3. There are four attributes (dimensions) A, B, C and D having 3, 2, 3 and 2 possible domain values (ranges), respectively. An ant starts from the "Start" vertex and constructs the antecedent part of the rule by adding conditions of the form attribute-value pairs (i.e., terms). After a term has been selected, all the other terms from the same attribute become prohibited for that ant. For instance, suppose the ant chooses C1 as its first term, it cannot select any

other terms pertaining to attribute C. Further, suppose that the ant subsequently chooses A3, D1 and B1, the terms that become prohibited after every move of the ant, are shown with the vertices being crossed in Fig. 3. The rule construction process is stopped when the ant reaches any of the vertices with the label "Stop". Due to the DAG like topology of search space, the constraints on the ant moves are very simple to understand and gives insights about why these constrains are added to the algorithm.

4.4. Complete algorithm

The working of the algorithm is based on the sequential learning strategy, given below. Basically, the algorithm discovers a rule that covers at least a few examples (equal to a user defined parameter '*n*') in the training set with the objective of high predictive accuracy. The examples that are covered by the rule are removed from the training set. The learning continues on the reduced training set until only a few examples (a user defined parameter) are left uncovered or all the examples are covered by the discovered rule list.

Repeat	
	Discover a *RULE* (with high predictive accuracy)
	Remove the training examples that are covered by the *RULE*
Until	(Only a few '*n*' training examples left uncovered/all examples are covered)

The core part of hmAntMiner-C is the outer most WHILE loop where several candidate rules are constructed based on sequential covering approach. It is mandatory for a predicted rule to cover at least a number of training samples equal to a user defined parameter *Min_Cases_Per_Rule* in order to handle the problem of over-fitting. The best rule within the discovered rule set is pruned and added to the discovered rule list. The training set is emptied of the samples covered by the discovered rule and then another rule is discovered (WHILE loop).

One best rule is discovered after one execution of the WHILE loop of Fig. 4. For the discovery of this rule, several candidate rules are constructed and evaluated based on certain criteria e.g., predictive accuracy. Class labels to the rule are assigned after construction of the rule antecedent part. During an iteration of the first inner REPEAT UNTIL loop of the algorithm, several ants are sent to construct antecedent part of the candidate rules (second inner REPEAT UNTIL loop). The quality of the candidate rules is determined after pruning and the iteration best rule is selected. This rule is used to update the pheromones of the trails in the antecedent construction graph (i.e., search space); where the pheromone values on the edges present in the rule are increased based on the quality of the rule and then later evaporation is performed on all the trails present in the search space (discussed later).

Each new ant in an iteration of the second inner REPEAT UNTIL loop is guided by the experience of the ants in previous iteration (of the first inner REPEAT UNTIL loop), available in the form of pheromone values. The best rule is updated if the iteration best rule is better than the best rule found so far. After several iterations of the first inner REPEAT UNTIL loop, the search of the swarm converges. The convergence is confirmed by analyzing that the iteration best rules are equivalent over iterations equal to a user defined parameter *no_rules_converge* or in other words, best rule is not updated during last few iterations equal to *no_rules_converge*. In this way, early convergence is possible before executing total number of iterations given as an input by user. The early convergence occurred most of the times as noticed in our experiments; thus depicts less dependency of the performance of algorithm based on the total number of iterations allowed to be executed and therefore makes the algorithm computationally less demanding. The learning process also stops if the size of the training set has reduced enough and contain training instances that are no more than a user defined parameter *Max_Uncovered_Cases*. For these uncovered cases (if any), a default rule (with empty antecedent) is added to the discovered rule list. Default rule covers all the remaining uncovered training samples and is assigned with the class labels based on certain criteria (e.g., class in majority within the uncovered samples). In the following subsections, the criterions used to handle different situations are discussed in fair amount of detail.

4.5. Pheromone initialization & search space reconstruction

The intelligent and self assisted moves of an ant are based on the pheromone and heuristic values associated with the edges in the search space. The edges in the search graph are initialized with the small pheromone values. As discussed earlier, hmAntMiner-C is a sequential learning algorithm; it learns a IF-THEN rule and removes the training instances covered by the rule from the training set. After every iteration of the main WHILE loop, the training set get reduces and used to reconstruct the antecedent construction graph, each time at the beginning of WHILE loop. It is therefore essential to apply pheromone initialization in the search space, each time it is reconstructed. The initial pheromone between two terms $term_i$ and $term_j$ belonging to two different attributes is computed as:

$$\tau_{ij}(iter = 1) = \frac{1}{\sum_{n=1}^{a} x_n b_n} \tag{7}$$

where a is the total number of attributes (excluding the class attribute), b_n is the number of possible values in the domain of an attribute A_n and x_n is set to 0 if the attribute A_n is that to which $term_i$ belongs, otherwise it is set to 1. The pheromones on the edges between terms of the same attribute are initialized with zero. Since all the pheromone values for the terms (competing to be added to the antecedent of the rule) are equivalent, the first ant has no historical information to guide its search process. This method of pheromone initialization has been adopted from our algorithm, AntMiner-C [28].

At the beginning of hmAntMiner-C, heuristic information plays an important role in leading the search process towards the right direction. Reconstructing the search space is one of the most important differences between hmAntMiner-C and the earlier versions of AntMiner. This reconstruction is based on the current reduced training set and therefore helps reducing the complexity and size of the search space, adaptively. When the best rule is discovered, the covered cases from the training set are removed. This would usually results in one or more than one term disappeared from the current reduced training set. The newly reconstructed search space also does not contain the disappeared term(s) and therefore incorporating this environment change in the search space, helps in reducing the computational complexity of the algorithm up to a significant level. Moreover, the terms whose cardinality in training set is less than *Min_Cases_Per_Rule* are never ever selected (This statement is valid for all the AntMiner versions proposed so far in the literature that use the parameter of *Min_Cases_Per_Rule*). These terms are also not added into the search space. The reason of doing this is very simple, "If we are not available with the information's about term(s) in the current training set, using that unavailable information for construing the rules is not sensible or helpful". These changes make the learning dynamics of ACO more efficient.

As an example, we have shown in Fig. 5(a), the number of terms (normalized between 0 and 1) in the search space after each iteration of the main WHILE loop during the execution of hmAntMiner and hmAntMiner-C for Pheno dataset (the only dataset used in the experiments with all the attributes being nominal-valued) using FunCat scheme (this experiment is performed on the same machine using the same parameter configuration). The other datasets used in the experiments are mixture of nominal

```
DiscoveredRuleList = {}; /* rule list is initialized as empty list */
TrainingSet = {all training samples};
WHILE( |TrainingSet| > Max_uncovered_samples) /*main WHILE loop (core of hmAntMiner-C)*/
    Construct the search space for reduced training set; /*adaptive reduction*/
    Calculate heuristic values for all the terms and pairs of terms;
    Initialize pheromone values on all the edges;
    iter = 1;          /* counter for iterations */
    best_rule = {};
    rcc = 1; /* counter for rule convergence test */
    REPEAT /*first inner REPEAT loop*/
        Iteration_best_rule = {};
        t = 1;    /* counter for ants */
        REPEAT /*second inner REPEAT loop*/
            Send Ant t to construct antecedent of the rule R_t;
            Prune rule R_t and asses its quality;
            IF (R_t is better than Iteration_best_rule)
                THEN Iteration_best_rule = R_t;
                IF (Iteration_best_rule is better than best_rule)
                    THEN best_rule = Iteration_best_rule;
                END IF
            END IF
            t = t + 1;
        UNTIL (t ≥ swarm_size)
        update pheromone of all the antecedent sub-graph trails based on Iteration_best_rule;
        IF (iter > 1 AND Iteration_best_rule == best_rule) /* update convergence test */
            THEN rcc = rcc + 1;
            ELSE rcc = 1;
        END IF
        iter = iter + 1;
    UNTIL (iter ≥ total_iterations) OR (rcc ≥ no_rules_converge)
    Prune the best best_rule and asses its quality;
    Add the pruned best_rule to DiscoveredRuleList;
    Remove the training samples covered by the pruned best_rule;
END WHILE
Add a default rule in the DiscoveredRuleList:
Prune the DiscoveredRuleList; (Optional)
```

Fig. 4. hmAntMiner-C Algorithm.

and continuous valued attribute. For these datasets, comparison (based on the number of terms) cannot be done in straight forward way, because hmAntMiner-C is not supported for continuous attributes. The execution time (normalized between 0 and 1) to find the rule set and area under precision-recall curve is shown in Fig. 5(b) and (c), respectively. More detail discussion regarding datasets used in the experiments and evaluation criteria based on precision-recall curve, is given in the following sub-sections. We can see in Fig. 5(a) that the number of terms after each iteration is effectively reducing for hmAntMiner-C (in turns reducing the size of search space) whereas it remain constant for hmAntMiner. The execution time and performance of hmAntMiner-C based on area under precision-recall curve (PRC) is also better than hmAntMiner.

4.6. Term selection

An ant incrementally add terms in the antecedent part of the rule that it is constructing during its tour in the search space. The selection of the next term is subject to the restriction that a term from its attribute A_n should not already be present in the current partial rule. In other words, once a term has been included in the rule then no other term from that attribute can be considered onwards. Subject to this restriction, the probability of selection of a term for addition in the current partial rule is given by the equation:

$$P_{ik}(t, iter) = \frac{\tau_{ik}{}^{\alpha}(iter)\eta_{ik}{}^{\beta}(s)}{\sum_{m=1}^{Total\_terms} x_m\{\tau_{im}{}^{\alpha}(iter)\eta_{im}{}^{\beta}(s)\}} \quad (8)$$

where $\tau_{ik}(iter)$ is the amount of pheromone associated with the edge between $term_i$ and $term_k$ for the current iteration equal to '*iter*', $\eta_{ik}(s)$ is the value of the heuristic function (for the edge between $term_i$ and $term_k$) for the current iteration s of the main WHILE loop (constant/similar for a batch of ants in 's'), x_m is a binary variable that is set to 1 if $term_m$ is selectable, otherwise it is set to 0. Selectable terms belong to the attributes which have not become prohibited. An attribute becomes prohibited if any of the terms associated with it, is already selected by the ant. The denominator is used to normalize the numerator value for all the possible choices.

The parameters α and β are used to control the relative importance of the pheromones and heuristic values for probability calculation. We use $\alpha = \beta = 1$, which means that we give equal importance to the pheromone and heuristic values. Eq. (8) is a classical equation and used in Ant System, MAX-MIN Ant System, Ant Colony System (where the state transition is also dependant on one other equation), AntMiner [14] (with $\alpha = \beta = 1$), and AntMiner+ [22].

4.7. Distance based correlated heuristic function

The heuristic value of an edge gives an indication of its usefulness and thus provides a basis to guide the search process. The heuristic function for antecedent construction graph is based on the correlation of the most recently chosen term (in the partial tour of the current ant) with other candidate terms currently available for selection. The two given terms $term_i$ and $term_k$ are strongly correlated if they minimize the variance of the set of examples (in the class space) covered by these two terms (by coverage we mean the examples that contain the given two terms). Our distance based correlated heuristic function is an extension to the heuristic used in CLUS-HMC and hmAntMiner where only single term is evaluated (i.e., no use of the ant current tour history). When the

Fig. 5. Comparison of hmAntMiner vs. hmAntMiner-C for pheno FunCat dataset based on: (a) number of terms after an iteration, (b) execution time and (c) area under PRC. (top-bottom).

learning is progressive, we can say that every single move of an ant in the search space, gives it a valuable experience towards learning the structure of underlying dataset. This experience regarding search space is propagated to the entire swarm by the means of pheromone. In order to make most reasonable (and possibly most accurate) next move, ants can see backward towards the moves it already made. This ensures that the next move would be appropriate to the current context rather than the context at the start of the search process, as was the case for hmAntMiner. For instance, the decision (generated by a brain) about how fast to run (or when to change direction), made by a Cheetah while preying a Deer is based on the context at that time and current position of the Cheetah rather than the position from where the Cheetah actually started his deadliest action. Same is the case for hill climbing and many of the other real world problems. With this motivation, we use history of the ant partial tour to guide it further in the search space. Only single recent term selected by an ant is considered for making a decision about the selection of next term within the search space. This makes the heuristic calculation computationally less demanding as compared to analyzing all the terms present in the current partial tour of an ant.

The association of 'n' class labels (present in the CHS) with a training sample is represented by a binary valued vector 'v', where $|v| = n$. The class label vector contain 1 at the ith position if i *th* class label in the CHS is associated with the given sample, otherwise the ith position value is set to 0. Given two class label vectors 'v_1' and 'v_2' for two samples, the distance between the vectors is defined as the weighted Euclidean distance, given below:

$$distance(v_1, v_2) = \sqrt{\sum_{i=1}^{n} w(l_i).(v_{1,i} - v_{2,i})^2} \quad (9)$$

where $w(l_i)$ is the weight associated with the ith class label, $v_{1,i}$ and $v_{2,i}$ are the values of the ith position of the class label vectors 'v_1' and 'v_2', respectively. The variance of the set of samples containing terms $term_i$ (most recently chosen term within the ant tour) and $term_k$ (the term we are evaluating to be added to the tour) is calculated as follows:

$$variance(Cov_{termi,termk}) = \frac{\sum_{j=1}^{|Cov_{termi,termk}|} distance(v_j, \hat{v})^2}{|Cov_{termi,termk}|} \quad (10)$$

where the $Cov_{termi,\,termk}$ is the set of samples covered by the terms: $term_i$ and $term_k$. $\hat{v}$ is the mean class label vector for the samples in $Cov_{termi,\,termk}$, such that the ith position value of this vector is calculated as follows:

$$\hat{v}_i = \frac{\sum_{j=1}^{|Cov_{termi,termk}|} v_{j,i}}{|Cov_{termi,termk}|} \quad (11)$$

The heuristic information of the $term_k$ (the term under evaluation by an ant to be selected next) in the context to the current position ($term_{is}$, most recently chosen term) of an ant is given as:

$$\eta_{termj} = \frac{variance(\max) - variance(Cov_{termi,termk})}{variance(\max)} \quad (12)$$

The higher the value of the heuristic function the minimum is the variance between the set of examples (covered by the two given terms) and the higher the attractive power of $term_k$ for selection. *variance* (max) is set equal to the sum of worst and best (minimum) variance values observed across all the terms (terms in pair). This setting of *variance* (max) makes it possible to assign a non-zero heuristic value to the worst term (giving a chance of selection to the worst term). It may please be noted that the heuristic calculations are done once at the beginning of the main WHILE loop for hmAntMiner-C algorithm (as the training examples only changes at end of main WHILE loop) and thus the computational complexity of algorithm becomes encouraging. The heuristic calculation on the edges between the 'Start' node (Start labeled term) and the first layer terms of the search space is a bit different. In this case, we ignore the Start labeled term. Now, $Cov_{Start,termk}$ represents the set of examples covered by only $term_k$.

In order to calculate distance between two class label vectors, we need to assign weights for each class labels in the CHS. The similarity between class labels at upper levels is important as these class labels are frequent and we are more confident about the correctness of similarity (at upper levels), due to availability of enough samples per class label. In other words, mistakes at upper level are discourage as these can lead to definite mistakes at lower levels. Keeping in view this concern, several weighting schemes are analyzed during the experiments of CLUS-HMC [12] algorithm. As per

their observations and findings, the preferred weighting scheme and the one used in hmAntMiner-C is given as follows:

$$w(l_i) = w_0 . \frac{\sum_{k=1}^{|parent\_l_i|} w(parent_k)}{|parent\_l_i|} \quad (13)$$

where $|parent\_l_i|$ represent(s) the number of parent class label(s) of the ith class label. $w(parent_k)$ is the weight of kth parent in the set $|parent\_l_i|$. $w_0$is the weight associated with the class labels at first level in the CHS (i.e., direct ancestor(s) of root class label) and arbitrarily set to 0.75 [12]. In this way, class labels at upper levels are assigned with high weight values as compared to the class labels deep in the hierarchy.

4.8. Rule quality

Once the rule is constructed by an ant, its quality is determined which is used to update the pheromone values in the search space. An appropriate quality measure is essential in order to influence the search process in right direction. The quality of the hierarchical multi-label classification rule is based on a measure proposed in hmAntMiner [8]. The samples in the current training set can be partitioned into two clusters based on the coverage of a rule. One of these clusters contain the samples satisfying all the conditions present in the rule antecedent part denoted as 'Cov_r'. Second set contain the samples, uncovered by the rule, represented as '$\sim Cov_r$'.

Quality of a rule is considered better if within cluster homogeneity of examples (in the class space) is high. In other words, same distance based variance measure can be applied again. The quality of a rule is attractive given that the intra-cluster variance gain (as defined above) of both the clusters is minimized as compared to the variance gain of entire training set. Thus the quality 'Q' of a rule 'r' corresponding to current train set 'Tr_Set' can be computed as:

$$Q_r = variance(Tr\_Set) - (variance(Cov_r) + variance(\sim Cover_r)) \quad (14)$$

The higher the quality of a rule (tour of an ant) the more the quantity of pheromone is concentrated over the parts (edges in the search space) between the conditions, (terms or vertices in the search space) present in the rule. The terms with high pheromone values are thus become more attractive for the ants in the subsequent iterations.

4.9. Rule pruning

Is it possible to further improve the quality of a discovered rule? One possibility is to apply the pruning process on the rule soon after it gets discovered (and before adding it to the discovered rule list). The pruning process is intended to improve the quality of the rule after removing irrelevant or copious terms and/or class labels from the antecedent and consequent parts of the rule, respectively. It also make the rules compact in size that adds to the comprehensibility power of the rules-based classification algorithms and which is attractive to the experts working in different domains.

The pruning procedure used in our work is a modified version of the pruning procedure proposed in AntMiner [14] and further investigated in hmAntMiner [8]. The entire process consists of several steps. At the first step, the quality of the rule is calculated using Q measure as per Eq. (14). The quality of the rule assists the pruning process in determining whether or not the pruning of the rule is effective during the successive steps of the pruning process. In the second step, the last term added to the antecedent part of the rule is iteratively removed using method remove_last_term(). This method removes the last term from the antecedent of the rule. The iterative procedure of the removal of last term from the antecedent continuous until the quality of the rule start decreases or a single term is left in the antecedent. Since, the coverage of the rule gets change at each such removal of the last term, probability values for the class labels are updated using Eq. (5), accordingly. It is important to note that the terms in the antecedent part of the rule is considered as an ordered list, and terms are removed in an order inverse to the order in which they were added to the rule [8] during the tour construction.

The algorithmic pseudo code for the pruning process is given in Fig. 6. It is assumed that *rule_r* be the rule that is required to be pruned. At the start of the pruning algorithm, a clone of *rule_r* is created and stored as *best_rule*. At each iteration of the repeat loop, a candidate rule *rule_i* is created by removing the last term from the antecedent of the *best_rule*. The *best_rule* is replaced with *rule_i*, given that the quality of *rule_i* has improved as compared to the quality of *best_rule*. The repeat loop is executed until either there is a single term left in the antecedent part of the *best_rule* or no improvement is detected for *rule_i* over the *best_rule*. After applying the pruning process, the pruned rule is returned as stored in the variable *best_rule*.

4.10. Pheromone update and evaporation

The decision of an ant about making it moves in the search space is controlled with two important parameters as shown in Eq. (8). Pheromone values on the edges are thus an important learning dynamic for the Ant Colony Optimization algorithm. The quality of the ant trails is used to make an efficient use of the pheromone values. The pheromone values on the edges constituting the trails are updated in proportion to the quality of the trails and thus define the learning directions for the subsequent transitions of the entire swarm.

For hmAntMiner-C, the pheromone values associated with the edges in antecedent search graph can be visualized as a square matrix having one dimension of the matrix equal to the number of total terms present in the current training set. The pheromone matrix is interpreted as an asymmetric matrix which helps in exploring the search space in great details. An example pheromone matrix is shown in Fig. 7. The pheromone matrix is asymmetric and captures the fact that pheromone values on edges originating from a term to other terms are kept the same in all the layers of the antecedent construction graph. In other words, the same matrix is valid for any pair of the two consecutive layers. The matrix is asymmetric because for a constructed rule with $term_i$ occurring immediately before $term_k$, the pheromone on edge between $term_i$ and $term_k$ ($\vec{t_i t_k}$) is updated but the pheromone on edge between $term_k$ and $term_i$ ($\vec{t_k t_i}$) remains unchanged. This is done to encourage exploration and discourage early convergence.

During the first iteration of the first inner repeat loop of the hmAntMiner-C algorithm, the pheromone values on the edges of the antecedent search graph are the ones initialized with the help of pheromone initialization procedure see e.g., Section 4.5. The pheromone values in the antecedent construction graph remain unchanged during the execution of the second inner repeat loop or in other words, all the ants have the same environment in terms of pheromone values. After completion of the second inner loop, the iteration best trail is used to update the pheromone matrix. In this way, the experience of the swarm in the iteration 't' of the first inner repeat loop is made available to the swarm in next iteration i.e., '$t + 1$' of the first inner repeat loop. It is assumed that each new iteration of the first inner repeat loop, results in a better experience of the swarm towards the problem search space given that learning is effectively progressing after employing the dynamics of ACO in an appropriate manner. The amount of pheromone on the edges

```
best_rule = rule_r;
q_best = Q (rule_r);

REPEAT /*first outer REPEAT loop*/
        rule_i = best_rule;
        rule_i→antecedent = remove_last_term ( rule_i→antecedent );
        q_i = Q (rule_i);
        rule_i→consequent = calculate_class_probability(rule_i→ antecedent );

        IF (q_i ≥ q_best) THEN
                best_rule = rule_i;
                q_best = q_i;
        END IF
UNTIL ( q_i < q_best OR |best_rule→antecedent| == 1)
```

Fig. 6. Pseudo code for rule pruning procedure.

	A1	A2	A3	B1	B2	C1	C2	C3	D1	D2
S	τ_{01}	τ_{02}	τ_{03}	τ_{04}	τ_{05}	τ_{06}	τ_{07}	τ_{08}	τ_{09}	τ_{010}

(a)

	A1	A2	A3	B1	B2	C1	C2	C3	D1	D2
A1	0	0	0	τ_{14}	τ_{15}	τ_{16}	τ_{17}	τ_{18}	τ_{19}	τ_{110}
A2	0	0	0	τ_{24}	τ_{25}	τ_{26}	τ_{27}	τ_{28}	τ_{29}	τ_{210}
A3	0	0	0	τ_{34}	τ_{35}	τ_{36}	τ_{37}	τ_{38}	τ_{39}	τ_{310}
B1	τ_{41}	τ_{42}	τ_{43}	0	0	τ_{46}	τ_{47}	τ_{48}	τ_{49}	τ_{410}
B2	τ_{51}	τ_{52}	τ_{53}	0	0	τ_{36}	τ_{37}	τ_{38}	τ_{39}	τ_{310}
C1	τ_{61}	τ_{62}	τ_{63}	τ_{64}	τ_{65}	0	0	0	τ_{69}	τ_{610}
C2	τ_{71}	τ_{72}	τ_{73}	τ_{74}	τ_{75}	0	0	0	τ_{79}	τ_{710}
C3	τ_{81}	τ_{82}	τ_{83}	τ_{84}	τ_{85}	0	0	0	τ_{89}	τ_{810}
D1	τ_{91}	τ_{92}	τ_{93}	τ_{94}	τ_{95}	τ_{96}	τ_{97}	τ_{98}	0	0
D2	τ_{101}	τ_{102}	τ_{103}	τ_{104}	τ_{105}	τ_{106}	τ_{107}	τ_{108}	0	0

(b)

Fig. 7. [28] The pheromone values for the antecedent search graph of example problem given in Fig. 3. The elements of (a) are the pheromone values on edges from Start node to nodes of the first layer of terms in the antecedent construction graph. The elements of (b) are pheromone values on the edges between the terms present in any of the two consecutive layers of the antecedent construction graph. For example, the first row elements are the pheromone values for edges from the term A1 to all the other terms in the next layer. These values are the same for edges between 1st and 2nd layer, 2nd and 3rd layer, and so on. If an ant chooses C1, A3, D1 and B1 terms in its trail, the elements τ_{06}, τ_{63}, τ_{39} and τ_{94} are updated according to Eq. (15). For normalization, each pheromone element in a row is divided with the summation of all pheromone values in the same row.

between consecutive terms occurring in the iteration best trail is updated according to the equation:

$$\tau_{ik}(t+1) = (1-\rho)\tau_{ik}(t) + (1 - \frac{1}{1+Q})\tau_{ik}(t) \quad (15)$$

where $\tau_{ik}(t)$ is the pheromone value encountered in iteration 't' (of the first inner repeat loop) between $term_i$ and $term_k$. The pheromone evaporation rate is represented by ρ and Q is the quality of the current iteration best trail.

The equation update pheromones by first evaporating a percentage of the previously seen pheromone and then adding a percentage of the pheromone dependant on the quality of the iteration best trail/rule. If the rule is well representative to the data available in the current training set, the pheromone added in quantity is greater than the pheromone evaporated and the terms constituting the rule become more attractive for the ants in the subsequent iterations. The evaporation in the equation improves exploration, otherwise in the presence of a static heuristic function the ants tend to converge quickly to the terms selected by the entire swarm during the first few iterations of the first inner repeat loop.

The next step is to normalize the pheromone values. Each pheromone value is normalized by dividing it with the summation of pheromone values on edges of all its competing terms. In the pheromone matrix (Fig. 7), normalization of the elements is done by dividing them with the summation of values of the row to which they belong. Note that for those rows where no change in the values has occurred after applying Eq. (15), the normalization process yields the same values as before. Referring to Fig. 3, normalization of pheromone value of an edge originating from a term is done by dividing it with the summation of all pheromone values of the edges originating from the same term. This process changes the amount of pheromone associated with those terms that do not occur in the iteration best rule but are the competitors of the selected terms.

4.11. Stopping criterions and default rule

In this section, we are going to discuss different criterions to stop the execution of hmAntMiner-C algorithm. A rule is restricted to cover at least a number of training instances equivalent to a user defined parameter *Min_Cases_Per_Rule*. This restriction helps in avoiding too specific rules that would cause over-fitting problem. Based on this restriction, there is a possibility that no rule is predicted during an iteration of the outer most while loop. This usually happens when the training set is reduced enough and contain very few instances and there in no term available to be added to the rule which covers instances equal to *Min_Cases_Per_Rule*. In such a situation, learning is stopped and a default rule with empty antecedent part is added to the discovered rule list which technically covers all the training instances. The class probability values for the default rule are calculated in the same way as discussed before with the only difference that the coverage of the default rule contains all the training instance present in the current reduced training set.

It is also important to note that the coverage of a rule in hmAntMiner-C has different meaning than that of the flat AntMiner's proposed in the literature. In flat AntMiner's, a rule is consider to cover a training sample, if the sample values is in accordance with the conditions of the rule antecedent and the class value of the sample also matches to the rule consequent part. It is however not straight forward to find a hierarchical rule with the consequent exactly matching with the class values of a training sample (based on a threshold value) and therefore for the coverage of a rule, consequent part is not checked against class values of training samples. Another such situation (for stopping the execution of algorithm) is faced when the size of the training set has reduced enough and contain training instances that are no more than a user defined parameter *Max_Uncovered_Cases.* In this case, a

Table 1
Detail of Bioinformatics datasets used in the experiments.

Class Hierarchy Structure	Dataset	# of samples in training set	# of samples in testing set	# of attributes	# of class labels
Tree (FunCat)	cellcycle	2, 476	1, 281	77	500
	desiri	2, 450	1, 275	63	500
	eisen	1, 587	837	79	462
	expr	2, 488	1, 291	551	500
	gash1	2, 480	1, 284	173	500
	pheno	1, 009	582	69	456
	seq	2, 580	1, 339	478	500
	spo	2, 437	1, 266	80	500
DAG (GO)	cellcycle	2, 473	1, 278	77	4, 126
	desiri	2, 447	1, 272	63	4, 120
	eisen	1, 583	835	79	3, 574
	expr	2, 485	1, 288	551	4, 132
	gash1	2, 477	1, 281	173	4, 126
	pheno	1, 005	581	69	3, 128
	seq	2, 568	1, 332	478	4, 134
	spo	2, 434	1, 263	80	4, 120

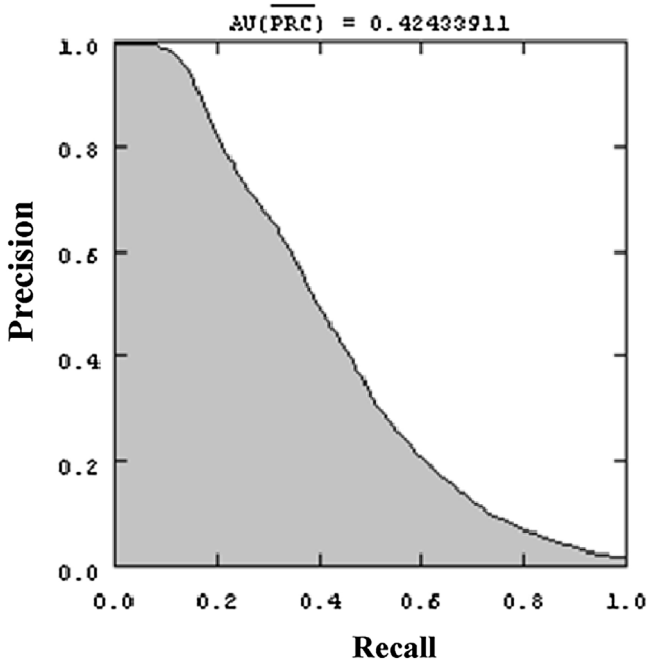

Fig. 8. PR curve generated for Pheno dataset with GO class hierarchical structure using hmAntMiner-C algorithm. For a lower (higher) value of recall there is usually a higher (lower) value of precision; showing the tradeoff between precision and recall. Shaded portion is the area under PR curve.

Table 2
Average number of class labels in hierarchy (CHS) and average number of labels per example.

All Datasets in	Avg. # of class labels	Avg. # of class labels per example
Tree (FunCat)	489	8.5
DAG (GO)	3, 932	34.2

default rule (with empty antecedent part) is added to the discovered rule list covering all the training instances; thus stopping the learning process of hmAntMiner-C algorithm.

4.12. Classifying unseen data and performance evaluation

After the algorithm finished its learning, the discovered rules are used (in order) to classify the unseen data examples. In particular, a test example is selected and given to the model learnt after training phase. The rule antecedent part is checked against the given test example, starting from the first predicted rule and continuing the search on the predicted rule list (in sequence), until a matching rule is found. At the end, consequent of the selected rule is assigned to the test example based on a threshold value. To be more specific, the class labels (present in the consequent of fired rule) having greater or equal probability values as compared to the threshold value, are predicted and assigned to the given test example.

Table 3
User-defined parameters used in experiments for hmAntMiner-C without any tuning based on optimization.

Parameter	Value
Number of Ants	30
Max. uncovered samples	10
Evaporation rate	0.15
No. of rules converged	10
Minimum cases per rule	10
Maximum iterations	1500
Alpha & Beta	1

Two common evaluation measures for a binary classification problem, from the domain of Information Retrieval are known as: precision and recall. Precision is the number of correctly classified positive examples (*TP*) divided by the number of examples labeled by the classifier as positive i.e., summation of *TP* (true positive) and *FP* (false positive). Precision represents the class agreement of the sample labels with the positive labels as predicted by the classifier. Recall shows the effectiveness of a classifier to identify positive examples and calculated by dividing the number of correctly classified positive examples with the number of positive examples present in the dataset i.e., summation of *TP* (true positive) and *FN* (false negative). The measures are defined as follows:

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN} \tag{16}$$

For performance evaluation of a hierarchical multi-label classification model, these measures can be extended (as done in CLUS-HMC and hmAntMiner) by micro-averaging the precision and recall values across all the class labels for a given threshold value '*t*' as follows:

$$\overline{Precision_t} = \frac{\sum_{i=1}^{n} TP_{i,t}}{\sum_{i=1}^{n} TP_{i,t} + \sum_{i=1}^{n} FP_{i,t}}, \qquad \overline{Recall_t} = \frac{\sum_{i=1}^{n} TP_{i,t}}{\sum_{i=1}^{n} TP_{i,t} + \sum_{i=1}^{n} FN_{i,t}} \tag{17}$$

Where $TP_{i,t}$, $FP_{i,t}$ and $FN_{i,t}$ are the number of true positives, false positives and false negatives predictions of the classifier for the *i*th class label in the CHS containing '*n*' class labels in total, for

the selected threshold value '*t*'. There is a tradeoff between precision and recall values for a particular threshold value. The lesser the threshold value the higher the chances that most of the class labels will be predicted (possibly, higher the recall values) while the proportion of correct prediction may increase or decrease (with tendency to decrease in precision values) [8] and vice versa. Deciding about a particular threshold value or a range of threshold values to be used during classifying unseen data and in turns for performance evaluation is totally an application oriented concern. As discussed in [8,12], different context in terms of applications, would require different threshold setting; we therefore use a threshold independent measure (known as area under PR curve) for performance evaluation of hmAntMiner-C algorithm, discussed as follows.

Precision-recall (PR) curves have been used recently in CLUS-HMC [8] and hmAntMiner [12] for the performance evaluation of hierarchical multi-label classification model. A point on the PR curve represents the precision of the model as a function of its recall value based on a threshold value. Decreasing the value of classification threshold '*t*' from 1.0 to 0.0, a number of pairs of precision-recall values are obtained that defines the overall PR curve, e.g., as shown in Fig. 8 for pheno dataset using GO scheme. The classification model performs better given that both the precision and recall values for the thresholds are high which in turn bends/shifts the PR curve towards the right-upper corner of the graph. The grey shaded area in the following figure is the area under PR curve denoted as AU($\overline{\text{PRC}}$) which is used to compare the classification algorithms for hierarchical multi-label classification task. The higher the value of the area under PR curve (with optimal = 1.0) the better is the performance of the algorithm. Some of the points on PR curve are interpolated [44] to approximate the area under PR curve by using the composite trapezoidal area approximation method between the set of points.

One of the great advantages of using PR curve for the performance evaluation of hierarchical multi-label classification problem is its capacity to deal with the highly skewed dataset [8,12]. As discussed earlier the class labels of a dataset organized using a class hierarchical structure (e.g., the ones used in the experiments of this work) tends to have many negative examples as compared to very small number of positive examples (for a class label) at the deeper levels of hierarchy. The ratio of negative to positive examples for a class node increases quickly as we move in the CHS from top to bottom thus posing an overly complex scenario for the classifiers that have problems with unbalanced datasets. Moreover, traditional classification techniques are lean to predict the class in majority as compared to the class in minority for a given unseen data sample [45]. It is therefore more important to correctly predict the positive class labels rather than rewarding the classifiers that correctly predict the negative samples in hierarchical multi-label classification (hmc) problems. This is what exactly the PR curve is reflecting that it does not consider the true negatives (*TN*) in the calculation of precision-recall points. This makes area under PR curve far more attractive choice for performance evaluation (for hmc) as compared to area under receiver operating characteristic (ROC) curve [12] that uses *TN* as well, in its calculation steps.

5. Results & discussion

In this section, we present the experimental results of our proposed method (hmAntMiner-C) in comparison with four other hierarchical multi-label classification algorithms. Three of the competitors are decision tree induction based algorithms [12]: 1) training a decision tree per each class label, called *CLUS-SC*, 2) inducing a decision tree in top-down fashion and at the same time, considering hierarchically consistent predictions only, called *CLUS-HSC* and 3) training a single global classification model predicting all the class labels in one go, called *CLUS-HMC* (source code and datasets [46]). Fourth competitor is an Ant Colony Optimization based hierarchical multi-label classification algorithm, called hmAntMiner [8]. The proposed algorithm (will be available online) is implemented in the Myra framework that is publically available (source code and datasets [47]) and written by Fernando E. B. Otero in Java language. Myra framework also include the implementation of hmAntMiner [8].

The hmAntMiner-C experiments are conducted on an Intel core i3 processor with a CPU clock rate of 2.4 GHz and 2 GB of main memory. This section is further divided in several subsections as follows. First, we discuss the datasets used in our experiments. Afterwards, in order to analyze statistical significance of the performances of all the competing algorithms, a nonparametric Friedman test with post-hoc Holm test [33,34] is conducted and differences are reported. At the end, overall model complexities (in terms of number of rules) of the algorithms are compared and statistically analyzed.

5.1. Datasets and evaluation methodology

In order to evaluate the proposed method, sixteen yeast datasets from Fernando et al. [8] are collected. Eight of these datasets have class hierarchical structure (CHS) organized with DAG structure (Gene Ontology) and for the remaining datasets the tree structure (FunCat) is used to define the CHS. These datasets include five different types of bioinformatics statistics, namely: sequence statistics, phenotype, secondary structure, homology, and expression; in order to describe different aspects of the gene in yeast genome. Datasets are briefly describe in follows, for more detail, readers are kindly referred to [12].

D1 (seq): keeps sequence statistics that rely on the amino acid sequence of the protein. The predictor attributes include: amino acid ratios, sequence, length, molecular weight and hydrophobicity. Most of the attributes of D1 are real valued, although some (like chromosome number or strand) are discrete.

D2 (pheno): comprises phenotype data, which denotes the growth or lack of growth of knock-out mutants that are absent the gene in question. The gene is disabled and the resulting organism is grown with a variety of media to determine what the modified organism might be sensitive or resistant to. The characteristics are discrete, and the data set is sparse.

Remaining datasets: The use of microarrays to record the expression of genes is popular in biology and bioinformatics. Microarray chips provide the means to test the expression levels of genes across an entire genome in a single experiment. Many expression data sets exist for yeast, and several of these were used. Characteristics for these data sets are real valued which describes the fold changes in expression levels.

From data mining point view, these datasets are too complex containing hundreds of attributes and thousands of instances and class labels where the labels are hierarchically structured (detail about datasets is given in Tables 1 & 2). In the experiments of hmAntMiner [8] and other three decision tree based competitors [12], 2/3 of each dataset was used as training and the remaining 1/3 for testing. For fair comparison, same training and testing set partitions are used for hmAntMiner-C experiments.

hmAntMiner-C is a stochastic algorithm, the results it would generate will be different if run multiple times on the same instance of a problem. We therefore run hmAntMiner-C 15 times (with different random seeds) for a dataset and then report the average of the generated output in terms of area under PRC and model size. Each run of hmAntMiner-C took on average 2.10 h (in the range: 0.22–6.09 h, excluding pheno dataset which took only 11 s) for FunCat based datasets and 11.34 h (in the range:

Table 4
Summary of the results in terms of AU($\overline{PRC}$) for the datasets used in the experiments (bold values represent best performing algorithm).

Class Hierarchy Structure	Dataset	hmAntMiner-C	hmAntMiner	CLUS-HMC	CLUS-HSC	CLUS-SC
Tree	cellcycle	0.154 ± 0.001	0.154 ± 0.001	0.172	0.111	0.106
(FunCat)	desiri	0.167 ± 0.001	0.161 ± 0.002	0.175	0.094	0.089
	eisen	0.175 ± 0.002	0.180 ± 0.003	0.204	0.127	0.132
	expr	0.167 ± 0.002	0.175 ± 0.002	0.210	0.127	0.123
	gash1	0.173 ± 0.002	0.175 ± 0.003	0.205	0.106	0.104
	pheno	0.163 ± 0.001	0.162 ± 0.001	0.160	0.152	0.149
	seq	0.166 ± 0.002	0.181 ± 0.002	0.211	0.091	0.095
	spo	0.167 ± 0.001	0.174 ± 0.002	0.186	0.103	0.098
DAG	cellcycle	0.430 ± 0.001	0.332 ± 0.002	0.357	0.371	0.252
(GO)	desiri	0.437 ± 0.001	0.334 ± 0.003	0.355	0.349	0.218
	eisen	0.450 ± 0.002	0.376 ± 0.002	0.380	0.365	0.270
	expr	0.440 ± 0.001	0.351 ± 0.003	0.368	0.351	0.249
	gash1	0.442 ± 0.001	0.356 ± 0.002	0.371	0.351	0.239
	pheno	0.427 ± 0.001	0.337 ± 0.001	0.337	0.416	0.316
	seq	0.450 ± 0.002	0.366 ± 0.003	0.386	0.282	0.197
	spo	0.441 ± 0.001	0.341 ± 0.003	0.352	0.371	0.213
Average:		**0.303**	0.260	0.277	0.235	0.178

Table 5
Summary of the results in terms of induced model size for all the datasets (bold values represent best performing algorithm).

Class Hierarchy Structure	Dataset	hmAntMiner-C	hmAntMiner	CLUS-HMC	CLUS-HSC	CLUS-SC
Tree	cellcycle	30.867 ± 1.606	28.667 ± 1.623	24	4037	9671
(FunCat)	desiri	07.000 ± 0.561	19.333 ± 1.661	4	3520	7807
	eisen	24.267 ± 1.926	19.000 ± 0.981	29	2995	6311
	expr	27.933 ± 1.987	30.600 ± 1.466	12	4711	10262
	gash1	21.400 ± 2.086	24.867 ± 1.701	10	4761	10447
	pheno	06.400 ± 0.335	07.400 ± 0.767	8	777	1238
	seq	17.467 ± 1.473	20.067 ± 1.152	14	4923	10443
	spo	7.7333 ± 0.740	15.800 ± 1.172	6	3623	8527
DAG	cellcycle	26.800 ± 1.642	35.400 ± 1.594	21	19085	36260
(GO)	desiri	09.533 ± 1.064	22.533 ± 1.939	10	16693	31175
	eisen	25.333 ± 2.072	18.200 ± 0.823	37	14384	24844
	expr	21.000 ± 1.670	28.600 ± 1.778	35	20812	38313
	gash1	21.733 ± 1.442	27.933 ± 0.918	30	20070	37838
	pheno	05.467 ± 0.307	07.133 ± 0.792	6	5691	6213
	seq	15.467 ± 1.490	18.067 ± 1.016	15	21703	38969
	spo	07.333 ± 0.760	26.333 ± 2.520	14	15552	35400
Average:		17.233	21.871	**17.188**	10208.563	19607.375

1.79–21.13 h, excluding pheno dataset which took only 45 s) for GO based datasets. Due to different configuration of hardware and resources used for experiments in [8,12], their reported execution time cannot meaningfully be compared with the one observed in this work.

There are six user defined parameters: swarm size, maximum uncovered samples, evaporation rate, convergence counter threshold, alpha and beta that is required to be settle for hmAntMiner-C. The values of these parameters are given in Table 3. These values have been chosen because they seems to provide reasonable performance as reported in the earlier versions in literature e.g., hmAntMiner [8]. The optimization of the parameter values is avoided since the goal is to evaluate the parameter settings generalization ability across a wide range of datasets used in the experiments. For a fair comparison, hmAntMiner-C uses the same parameter values as reported for hmAntMiner with the same number of total runs (i.e., fifteen) to generate the average results.

5.2. Experimental results over datasets

In this section, the results of hmAntMiner-C are compared with the four competitors as mentioned before on the basis of area under PRC (denoted as AU($\overline{PRC}$)) and model size (in terms of number of rules present in the classification model). For *CLUS-HMC*, *CLUS-HSC*, and *CLUS-SC* the model size represents the number of leaf nodes in the induced decision tree which is essentially a rule (a conditional path from root node to the child node) [8]. Since these decision tree based classifiers are deterministic algorithms (means, they generate the same model each time they are train on the same given data), they are run just once to generate the output performance values.

For hmAntMiner-C and hmAntMiner results, a single value in Tables 4 & 5 corresponds to the average value obtained across the total runs followed by the standard deviation in the form (average ± standard deviation). Table 4 present the results concerning the area under PRC values where the higher the values the better is the discrimination power of the classifier. Based on the average area under PRC across all the datasets, proposed algorithm (with avg.AU($\overline{PRC}$) = 0.303) performs better than its competitors. In Table 5, the results are summarized considering the size of classification model, where the smaller the values the simpler and compact is the classification model; thus attractive for domain experts in terms of comprehensibility and validity of the model. Fig. 9 illustrates a sample of precision–recall curves of hmAntMiner-C, hmAntMiner, and Clus- HMC for 'cellcycle', 'pheno' and 'seq' FunCat data sets and 'seq', 'eisen' and 'pheno' Gene Ontology data sets.

Based on the classification model size, CLUS-HMC performs the best with avg. model size = 17.188. It can be observed from the results summarized in Table 4 that CLUS-HMC performs better for FunCat datasets (excluding pheno dataset where proposed algorithm generates higher AU($\overline{PRC}$) value). On the other hand, for all

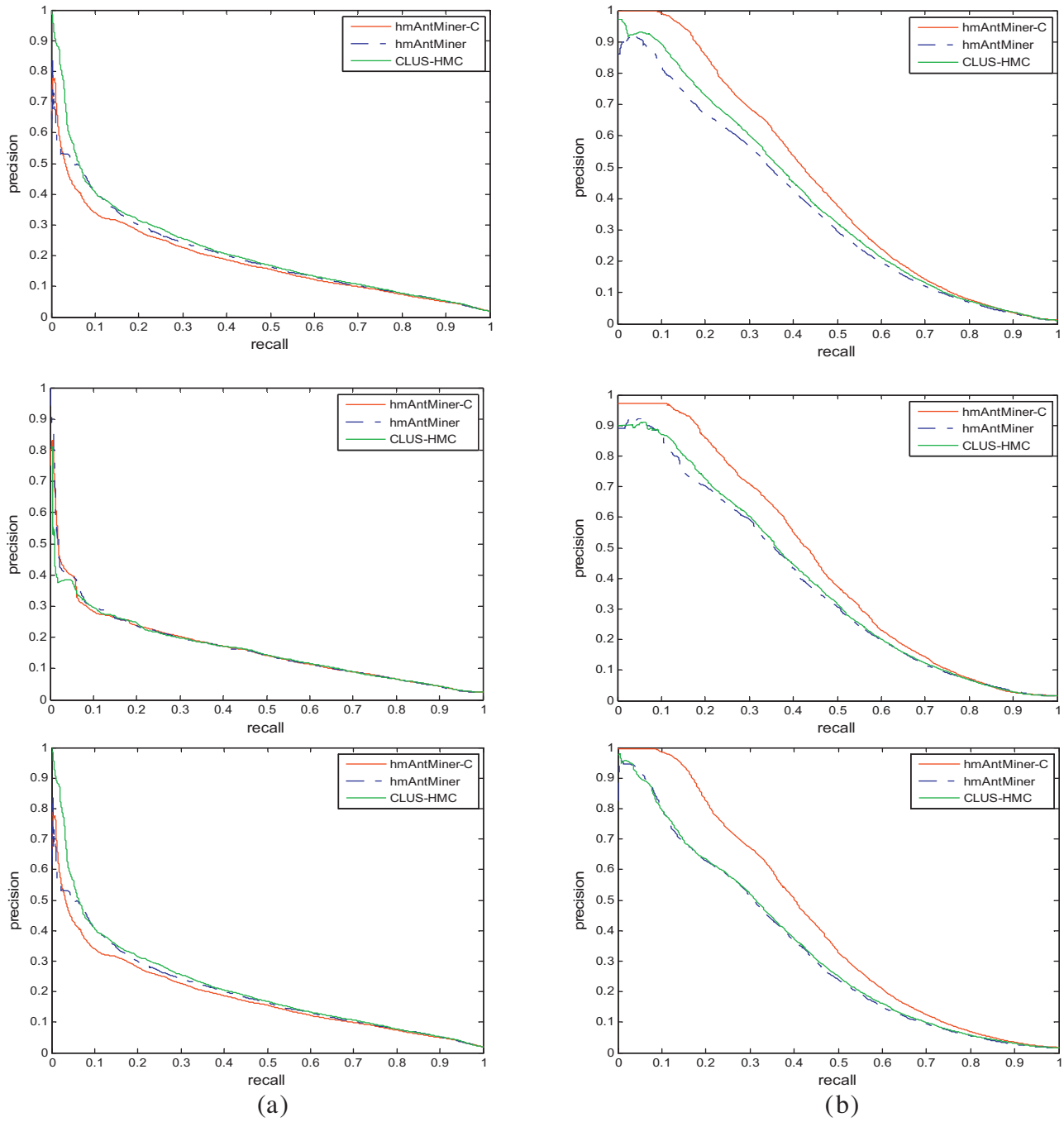

Fig. 9. A sample of precision–recall curves of hmAnt-Miner, Clus-HMC, and hmAnt-Miner for: a) 'cellcycle', 'pheno' and 'seq' FunCat data sets (top-bottom) and b) 'seq', 'eisen' and 'pheno' Gene Ontology data sets (top-bottom) where the closest to the upper-right-corner, the better (more accurate the model) the curve.

GO datasets, proposed algorithm is the best performing strategy. Keeping in view this trend in the results, we are going to analyze whether these performance differences are actually statistically significant or not?, based on three configurations of datasets: 1) for FunCat datasets only, 2) for GO datasets only, and 3) combined (both for FunCat & GO datasets). Discussion about statistical analysis is given in following paragraphs.

In order to analyze whether a specific algorithm performs significantly better than the others, a non-parametric statistical test (Friedman) is employed. The Friedman test is chosen because it does not make any assumptions about the distribution of the underlying data and a recommended and suitable test to compare a set of classifiers over multiple data sets, according to the guidelines presented in [33,34]. Table 6 presents the summary of the comparisons of the best algorithm (the algorithm with the best average rank, considered as *control* algorithm) with the remaining algorithms according to the non-parametric Friedman test with the Holm's post-hoc test [33,34] in terms of predicative accuracy (AU($\overline{\text{PRC}}$)) and model size. For each algorithm, the average rank (the lower the average rank the better the algorithm's performance), the *p-value* (when the average rank is compared to the average rank of the algorithm with the best rank i.e., control algorithm) and *Holm* critical value obtained by Holm's post-hoc test are reported. Based on the fact that the *p-value* is lower than the critical value (at 1% significance level), the values in a row are shown in bold when there is a significant difference between the average ranks of an algorithm

Table 6
Summary of the comparisons of the algorithms based on (i) predictive accuracy and (ii) model size, according to the non-parametric Friedman test with the Holm's post-hoc test on 0.01 significance level (bold values represent the significant difference between control algorithm and corresponding algorithm in the row).

Comparison on the basis	Dataset Configuration	Algorithm	Avg. rank	p	Holm
Predictive accuracy i.e., AU($\overline{PRC}$)	GO	hmAntMiner-C (control)	1.0000	–	–
		CLUS-HMC	2.4375	0.0690	0.05
		CLUS-HSC	**3.0625**	**0.0090**	**0.025**
		hmAntMiner	**3.5000**	**0.0016**	**0.0167**
		CLUS-SC	**5.0000**	**4.2004E-7**	**0.0125**
	FunCat	CLUS-HMC (control)	1.2500	–	–
		hmAntMiner	2.1875	0.2357	0.05
		hmAntMiner-C	2.5625	0.0969	0.025
		CLUS-HSC	**4.2500**	**1.4780E-4**	**0.0167**
		CLUS-SC	**4.7500**	**9.5469E-6**	**0.0125**
	Combined (FunCat + GO)	hmAntMiner-C (control)	1.7813	–	–
		CLUS-HMC	1.8438	0.9110	0.05
		hmAntMiner	2.8438	0.0573	0.025
		CLUS-HSC	**3.6563**	**7.9623E-4**	**0.0167**
		CLUS-SC	**4.8750**	**3.1253E-8**	**0.0125**
Model size	GO	hmAntMiner-C (control)	1.3750	–	–
		CLUS-HMC	2.1250	0.3428	0.05
		hmAntMiner	2.5000	0.1547	0.025
		CLUS-HSC	**4.0000**	**8.9891E-4**	**0.0167**
		CLUS-SC	**5.0000**	**4.5332E-6**	**0.0125**
	FunCat	CLUS-HMC (control)	1.5000	–	–
		hmAntMiner-C	2.0000	0.5271	0.05
		hmAntMiner	2.5000	0.2059	0.025
		CLUS-HSC	**4.0000**	**0.0016**	**0.0167**
		CLUS-SC	**5.0000**	**9.5469E-6**	**0.0125**
	Combined (FunCat + GO)	hmAntMiner-C (control)	1.6875	–	–
		CLUS-HMC	1.8125	0.8230	0.05
		hmAntMiner	2.5000	0.1461	0.025
		CLUS-HSC	**4.0000**	**3.5230E-5**	**0.0167**
		CLUS-SC	**5.0000**	**3.1120E-9**	**0.0125**

Table 7
Summary of all pairwise comparisons according to the non-parametric Friedman test with the Holm's post-hoc test in terms of (i) predictive accuracy and (ii) model size, according to the non-parametric Friedman test with the Holm's post-hoc test on 0.01 significance level (bold values represent significant difference between two competing techniques).

Comparison on the basis of	Hypothesis	p_{Holm} (Combined)	p_{Holm} (FunCat)	p_{Holm} (GO)
Predictive acc.	hmAntMiner-C vs. CLUS-HMC	0.9110	0.3875	0.2889
	hmAntMiner-C vs. hmAntMiner	0.2294	1.0541	0.0125
	hmAntMiner-C vs. CLUS-HSC	**0.0056**	0.1640	0.0636
	hmAntMiner-C vs. CLUS-SC	**3.1253E-7**	0.0396	**4.200E-6**
	CLUS-HMC vs. hmAntMiner	0.2294	0.7070	0.5369
	CLUS-HMC vs. CLUS-HSC	**0.0071**	**0.0013**	0.8584
	CLUS-HMC vs. CLUS-SC	**5.2904E-7**	**9.5470E-5**	0.0107
	hmAntMiner vs. CLUS-HSC	0.2922	0.0545	0.8584
	hmAntMiner vs. CLUS-SC	**0.0022**	**0.0095**	0.2889
	CLUS-HSC vs. CLUS-SC	0.1462	1.0541	0.0855
Model size	hmAntMiner-C vs. CLUS-HMC	0.8230	1.0541	0.6856
	hmAntMiner-C vs. hmAntMiner	0.4383	1.0541	0.6189
	hmAntMiner-C vs. CLUS-HSC	**2.4660E-4**	0.0684	**0.0071**
	hmAntMiner-C vs. CLUS-SC	**3.1120E-8**	**0.0013**	**4.533E-5**
	CLUS-HMC vs. hmAntMiner	0.4383	0.8236	0.6856
	CLUS-HMC vs. CLUS-HSC	**5.4667E-4**	0.0125	0.1062
	CLUS-HMC vs. CLUS-SC	**1.0659E-7**	**9.5469E-5**	**0.0024**
	hmAntMiner vs. CLUS-HSC	0.0364	0.2889	0.2889
	hmAntMiner vs. CLUS-SC	**6.1954E-5**	0.0125	0.0109
	CLUS-HSC vs. CLUS-SC	0.2945	0.8236	0.6189

and the control algorithm, and it shows that the control algorithm has significantly outperformed the counterpart algorithm in that row.

According to the statistics of Table 6, hmAntMiner-C (proposed algorithm) is the best performing algorithm both in terms of predictive accuracy and model size with the lowest average rank, for GO and combined (overall) dataset configuration. For FunCat datasets, CLUS-HMC stood first for both the evaluation measures. Based on predictive accuracy, the proposed algorithm has statistically significantly defeated three algorithms (hmAntMiner, CLUS-HSC and CLUS-SC) for GO datasets. Whereas for FunCat and combined (overall) datasets, it has performed statistically significantly better than CLUS-HSC and CLUS-SC algorithms in terms of predictive accuracy as well as model size. CLUS-HMC is a strong competitor and never defeated significantly in any single case. CLUS-SC is the worst performing strategy with poor predictive accuracy rate and generates a very huge model size, thus assigned with the largest average ranking in all the cases. With the statistics in Table 6, we can say that the proposed algorithm gives best result (for overall datasets) by predicting the class labels more accurately and yielding the compact classification model sizes, thus become an attractive choice for hierarchical multi-label classification task.

Table 7 presents the pairwise comparisons between all the algorithms based on the significance level of 0.01 for all the experimental setups given in Tables 4 & 5 (in terms of predictive accuracy and model size). For each hypothesis (pair of algorithms), the adjusted *p-value* using Friedman test followed by Holm's post-hoc test is reported and significant differences are shown in bold. Considering the statistics given in Table 7, the proposed algorithm has performed statistically significantly better when compared with the CLUS-HSC and CLUS-SC algorithms based on predictive accuracy and model size for combined dataset configuration. The proposed algorithm is also significantly better than CLUS-HSC and CLUS-SC algorithms based on model size and better than CLUS-SC based on predictive accuracy for GO datasets. CLUS-HMC and hmAntMiner are shown to be strong competitors to our algorithm and never defeated significantly based on the pairwise comparisons. There are other highlighted entries in Table 7 which shows that the differences between the competing algorithms (either based on predictive accuracy or model size) are significant and the algorithm with the low average rank as given in Table 6 for the corresponding experiment can be reported as winner.

The empirical evaluation have shown that the proposed algorithm is competitive to CLUS-HMC and hmAntMiner and assigned with the best average ranking based on the results generated for all the datasets. The proposed algorithm results in simpler classification model with high predictive performance and therefore can attract the experts working in different domains. As mentioned in [8], all (CLUS) decision tree induction algorithms work based on the predictive clustering trees (PCT) framework which is evolved due to prolong efforts over a period of more than a decade. Considering the immature level of Ant Colony Optimization based classification algorithms as compared to CLUS decision trees, the results are encouraging; as our algorithm is the second attempt to tackle hierarchical multi-label classification task following the hmAntMiner [8] and even then offers promising results.

6. Conclusion

In this article, we have presented a novel Ant Colony Optimization based hierarchical multi-label classification algorithm, named hmAntMiner-C. A fair amount of discussion about different types of hierarchical classification problems and different categories of corresponding solutions is also provided to enhance the understanding regarding the target problem. Extending on the ideas of our previous flat classification algorithm AntMiner-C, the hmAntMiner-C is tailor to handle the hierarchical multi-label classification task with both: tree (a class node can have single parent apart from root) and DAG (a class node can have multiple parents apart from root) class hierarchical structures. hmAntMiner-C employs a sequential covering approach and discover a single global classification model in the form of IF-THEN rules that are used in sequence during the testing phase. The usage of a new correlation based heuristic which takes into consideration the relationships between attribute-value pairs and distances in the class space, makes the proposed algorithm very well suited to the underlying problem.

The proposed algorithm is evaluated on sixteen bio-informatics datasets related to protein function prediction, containing hundreds of attributes and thousands of instances and class labels that are organized based on tree and DAG class hierarchical structure. The empirical comparisons are performed with three decision tree induction based algorithms and an ACO based hierarchical multi-label classification algorithm. According to the experimental results, hmAntMiner-C is significantly superior to the competitors CLUS-HSC and CLUS-SC (based on several statistical tests) in terms of prediction accuracy with simplistic model complexities. hmAntMiner-C is assigned with the best ranking based on its remarkable performance. We however concluded that the other two competitors (CLUS-HMC and hmAntMiner) have also performed up to the mark and therefore cannot be ignored for the problem being tackled in this work.

There are several future avenues in order to extend the proposed technique. First, it would be interesting to equip the algorithm with a dynamic continuous variable handling mechanism, that would give more insights into the data, being mined. It might also be useful for reducing the size of classification model built by algorithm and avoid the over fitting problem which would make the algorithm more robust. It is also encouraging to investigate the variations of new rule pruning strategies or new rule evaluation measures. A new sequential covering approach introduced in cAntMiner-PB [37] is worth enough to be investigated for optimizing the quality of the discovered rules.

References

[1] Yen-Liang Chen, Hsiao-Wei Hu, Kwei Tang, Constructing a decision tree from data with hierarchical class labels, Expert Syst. Appl. 36 (No. 3) (2009) 4838–4847.
[2] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.
[3] J.R. Quinlan, Generating production rules from decision trees, in: Proc. Int. Joint Conf. Artificial Intelligence, San Francisco, USA, 1987.
[4] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.
[5] A. Freitas, A. deCarvalho, A tutorial on hierarchical classification with applications in bioinformatics, in: D. Taniar (Ed.), Research and Trends in Data Mining Technologies and Applications, 2007, pp. 175–208.
[6] The gene ontology consortium: gene ontology: tool for the unification of biology, Nat. Genet. 25 (2000) 25–29.
[7] F.E.B. Otero, A.A. Freitas, C.G. Johnson, A hierarchical classification ant colony algorithm for predicting gene ontology terms, in: Proc. of the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBio), LNCS 5483, Springer, 2008, pp. 68–79.
[8] F.E.B. Otero, A.A. Freitas, C.G. Johnson, A hierarchical multi-label classification ant colony algorithm for protein function prediction, Memet. Comput. 2 (3) (2010) 165–181.
[9] F.E.B. Otero, New Ant Colony Optimization algorithms for hierarchical classification of protein functions, in: PhD Thesis, 2010.
[10] C.N. Cilla, B. Becker, Alex A. Freitas, A survey of hierarchical classification across different application domains, Data Min. Knowl. Discov. 2 (2010) 31–72.
[11] D. Koller, M. Sahami, Hierarchically classifying documents using very few words, Proc. of the 14th Int'l Conf. on Machine Learning (1997) 170–178.
[12] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, H. Blockeel, Decision trees for hierarchical multi-label classification, Mach. Learn. 73 (No. 2) (2008) 185–214.
[14] R.S. Parpinelli, H.S. Lopes, A.A. Freitas, Data mining with an ant colony optimization algorithm, IEEE Trans. Evol. Comput. 6 (No. 4) (2002) 321–332.
[15] F. Otero, A. Freitas, C. Johnson, cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes, in: Ant Colony Optimization and Swarm Intelligence (ANTS), LNCS 5217, Springer, 2008.
[16] A.P. Engelbrecht, Computational Intelligence, An Introduction, 2nd edition, John Wiley & Sons, 2007.
[17] A.P. Engelbrecht, Fundamentals of Computational Swarm Intelligence, John Wiley & Sons, 2005.
[18] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann/Academic Press, 2001.
[19] M. Dorigo, T. Stützle, Ant Colony Optimization, MIT Press, Cambridge, MA, 2004.
[20] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybernat. 26 (February, Part B (No. 1)) (1996).
[21] M. Dorigo, L.M. Gambardella, Ant colony system: a cooperative learning approach to the travelling salesman problem, IEEE Trans. Evol. Comput. 1 (No. 1) (1997).
[22] D. Martens, M. de Backer, R. Haesen, J. Vanthienen, M. Snoeck, B. Baesens, Classification with ant colony optimization, IEEE Trans. Evol. Comput. 11 (No. 5) (2007).
[23] A. Abraham, C. Grosan, V. Ramos, Swarm intelligence in data mining Studies in Computational Intelligence, vol. 34, Springer, 2006, 2017.
[24] J. Han, M. Kamber, Data Mining: Concepts and Techniques, 2nd ed., Morgan Kaufmann Publishers, 2006, 2017.
[25] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd ed., Morgan Kaufmann, 2005.
[27] S. Khan, M. Bilal, M. Sharif, M. Sajid, R. Baig, Solution of n-Queen problem using ACO, in: International Multitopic Conference, Islamabad, IEEE, 2009, pp. 1–5.

[28] A.R. Baig, W. Shahzad, A correlation based AntMiner for classification rule discovery, Neural Comput. Appl. J. (2010) (in press and available online from Springer website for NCA journal).
[29] S. Kiritchenko, S. Matwin, A.F. Famili, Functional annotation of genes using hierarchical text categorization, BioLINK SIG: Linking Literature, Inf. Knowl. Biol. (2005).
[33] S. García, F. Herrera, An extension on 'Statistical Comparisons of Classifiers over Multiple Data Sets' for all pairwise coparisons, Mach. Learn. Res. 9 (2008) 2677–2694.
[34] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Mach. Learn. Res. 7 (2006) 1–30.
[37] F.E.B. Otero, A.A. Freitas, C.G. Johnson, A new sequential covering strategy for inducing classification rules with ant colony algorithms, IEEE Trans. Evol. Comput. (2012) (to appear).
[38] H.W. Mewes, K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, D. Fr-ishman, MIPS: a database for protein sequences and complete genomes, Nucl. Acids Res. 27 (1999) 44–48.
[39] J. Rousu, C. Saunders, S. Szedmak, J. Shawe-Taylor, Kernel-based learning of hierarchical multilabel classification models, J. Mach. Learn. Res. (2006) 1601–1626.
[40] A. Clare, A. Karwath, H. Ougham, R. King, Functional bioinformatics for Arabidopsis thailana, Bioinformatics 22 (No. 9) (2006) 1130–1136.
[41] N. Holden, A. Freitas, Improving the performance of hierarchical classification with swarm intelligence, LNCS, in: Proc. of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, 973, 2008, pp. 48–60.
[42] A. Secker, M. Davies, A. Freitas, J. Timmis, M. Mendao, D. Flower, An experimental comparison of classification algorithms for the hierarchical prediction of protein function, 3rd UK Knowledge Discovery and Data Mining Symposium (2007) 13–18.
[43] N. Holden, A. Freitas, Hierarchical classification of protein function with ensembles of rules and particle swarm optimisation, Soft Comput. 13 (No. 3) (2009) 259–272.
[44] J. Davis, M. Goadrich, The relationship between precision-recall and roc curves, Proc. of the 23rd Inter-national Conference on Machine learning, ACM (2006) 233–240.
[45] C. Seiffert, T.M. Khoshgoftaar, J. Van Hulse, A. Napolitano, RUSBoost: Improving classification performance when training data is skewed, Proc. of 19th International Conference on Pattern Recognition, IEEE (2008) 1–4.
[46] http://dtai.cs.kuleuven.be/clus/hmcdatasets.
[47] http://sourceforge.net/projects/myra/files/.
[48] Abdul Rauf Baig, Waseem Shahzad, Salabat Khan, Correlation as a heuristic for an accurate and comprehensible ACO based classifier, IEEE Trans. Evol. Comput. 17 (Issue 5) (2012) 686–704.
[49] Salabat Khan, Abdul Rauf Baig, Waseem Shahzad, Abdul rauf baig, waseem shahzad, a novel ant colony optimization based single path hierarchical classification algorithm for predicting gene ontology, Appl. Soft Comput. 16 (2014) 34–49.