Applied Soft Computing Journal 77 (2019) 584–604

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

Inducing Hierarchical Multi-label Classification rules with Genetic Algorithms

Ricardo Cerri [a,*], Márcio P. Basgalupp [b], Rodrigo C. Barros [c], André C.P.L.F. de Carvalho [d]

[a] Department of Computer Science, Federal University of São Carlos, Brazil
[b] Instituto de Ciência e Tecnologia, Universidade Federal de São Paulo, Brazil
[c] School of Technology, Pontifícia Universidade Católica do Rio Grande do Sul, Brazil
[d] Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Brazil

HIGHLIGHTS

- A Genetic Algorithm for Hierarchical and Multi-label Classification is proposed.
- A sequential covering procedure is used to generated a set of classification rules.
- Several variations of the algorithm are proposed and evaluated.
- Performance is evaluated using hierarchical protein function prediction datasets.

ARTICLE INFO

Article history:
Received 7 February 2018
Received in revised form 8 January 2019
Accepted 11 January 2019
Available online 1 February 2019

Keywords:
Hierarchical Multi-label Classification
Protein function prediction
Machine learning
Genetic Algorithms
Rule induction

ABSTRACT

Hierarchical Multi-Label Classification is a challenging classification task where the classes are hierarchically structured, with superclass and subclass relationships. It is a very common task, for instance, in Protein Function Prediction, where a protein can simultaneously perform multiple functions. In these tasks it is very difficult to achieve a high predictive performance, since hundreds or even thousands of classes with imbalanced data distributions have to be considered. In addition, the models should ideally be easily interpretable to allow the validation of the knowledge extracted from the data. This work proposes and investigates the use of Genetic Algorithms to induce rules that are both hierarchical and multi-label. Several experiments with different fitness functions and genetic operators are preformed to obtain different Hierarchical Multi-Label Classification rules. The different proposed configurations of Genetic Algorithms are evaluated together with state-of-the-art methods for HMC rule induction based on Ant Colony Optimization and Predictive Clustering Trees, using many datasets related to the Protein Function Prediction task. The experimental results show that it is possible to recommend the best configuration in terms of predictive performance and model interpretability.

1. Introduction

Most classification tasks consist of assigning a single class to a given instance. Moreover, those problems consider a flat structure of classes, with no hierarchical relationship among them. However, in many real-world applications, there is a hierarchical structure that should be considered, since the classes involved can be specialized into subclasses or grouped into superclasses. These tasks are in turn known in the Machine Learning (ML) literature as Hierarchical Classification (HC). Depending on the class relationships, the hierarchical structure can be either in the form of a tree or of a directed acyclic graph (DAG).

While in trees a node can have only one parent class – thus there is only one possible path between a class and the tree root – in a DAG structure a node can have more than one parent class, generating multiple possible paths between classes and root node.

An HC task can be more complex when an instance can be simultaneously assigned to two or more hierarchical paths, which means an instance can be classified as belonging to more than one class within the same hierarchical level. HC tasks that follow such a description are known as Hierarchical Multi-Label Classification (HMC) problems, which can be formally defined as follows:

Given an input space $\mathbf{X}$ *and a set of classes* C*, find a function (classifier)* f *capable of classifying each instance* $\mathbf{x}_i \in \mathbf{X}$ *into a set of classes* $C_i \in C$*. The classifier* f *should not violate the constraints dictated by the class hierarchy while maximizing a quality criterion or minimizing a loss function* q*.*

Regarding the so-called constraints of the hierarchy, we assume that all superclasses of a given class should be predicted for the

* Corresponding author.
E-mail addresses: cerri@ufscar.br (R. Cerri), basgalupp@unifesp.br (M.P. Basgalupp), rodrigo.barros@pucrs.br (R.C. Barros), andre@icmc.usp.br (A.C.P.L.F. de Carvalho).

https://doi.org/10.1016/j.asoc.2019.01.017
1568-4946/

case in which the class was predicted. The quality criterion to be optimized, q, can be any given predictive performance measure. Such a measure can consider, e.g., the distance between predicted and true classes within the hierarchy (possibly measured as the number of edges between them). This measure is naturally based on the premise that closer classes within the hierarchy are more similar. Hence, incorrect classifications may be weighted proportionally to the number of edges between true and predicted classes. The larger the number of edges, the larger the classification error.

Two main approaches, namely local and global, are frequently used to deal with HMC tasks. While the local approach divides the original task into many subtasks (each assigned to a different classifier), the global approach induces a single classifier to deal with all classes at once. The local approach can be further divided depending on how local information is used during training. According to [1], three main strategies are used: one **l**ocal **c**lassifier per **n**ode (LCN), one **l**ocal **c**lassifier per **p**arent **n**ode (LCPN), and one **l**ocal **c**lassifier per **l**evel (LCL). The LCN strategy induces one binary classifier for each class of the hierarchy [2]. The LCPN strategy induces a multi-class classifier for each internal class to distinguish between its subclasses [3]. The LCL strategy induces one multi-label classifier for each hierarchical level, each classifier responsible for the prediction in its associated level [4]. There are very few cases in which a hybrid local+global approach is employed in an attempt to extract the best of both worlds [5,6].

A typical application of HMC methods is Protein Function Prediction (PFP), because proteins often perform many functions that are hierarchically organized. Proteins perform almost all functions related to cell activity, such as biochemical reactions, cell signaling, structural, and mechanical functions [7]. In the PFP task, both local, global, and hybrid methods have been applied [5,6,8–12].

Among the HMC methods proposed in the literature, very few are able to generate interpretable models, i.e., to generate a set of classification rules. To the best of our knowledge, there are only two studies in the literature, based on either Decision Trees Induction [13] or on Ant-Colony Optimization [9]. Thus, there is clearly a lack of methods focusing both on high predictive performance and on model interpretability. In this direction, we propose the investigation of a Genetic Algorithm (GA) to induce HMC rules for PFP. More specifically, we propose a method called Hierarchical Multi-label Classification with a Genetic Algorithm (HMC-GA), which thoroughly extends our previous work [14]. HMC-GA employs a sequential covering procedure to evolve antecedents of HMC rules. This work extends HMC-GA implementing novel fitness functions and genetic operators to evolve rules with relational and propositional tests. Also, we show results with a different class hierarchy, structured as a tree. The HMC-GA extensions implemented in this work are listed next.

- Evolution of three different types of rules:
  1. Traditional rules that only evaluate if an attribute value A_k satisfies a test condition, *e.g.* $A_k \leq x_{i,k}$. These tests are known as propositional tests;
  2. Rules that only compare the values of different attributes, *e.g.* $A_1 \leq A_2$. These tests are called relational tests [15];
  3. Rules with both propositional and relation tests;
- Experiments with tree different fitness functions:
  1. Variance gain;
  2. Weighting the variance gain and percentage of covered instances;
  3. Weighting of variance gain and area under the precision–recall curve;
- Experiments with four variations regarding the crossover procedure:
  1. Uniform crossover;
  2. Uniform crossover with local search;
  3. Distance-based uniform crossover;
  4. Distance-based uniform crossover with local search;

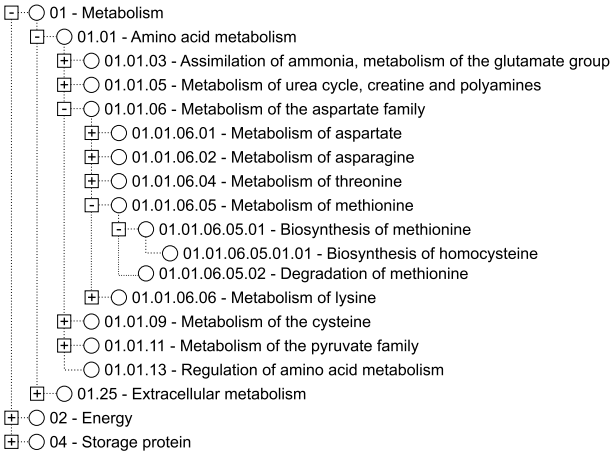


Fig. 1. Part of the Funcat hierarchical taxonomy. *Source:* Adapted from www.helmholtz-muenchen.de/en/ibis.

Based on combinations of all the previously-listed configurations (rules × fitness functions × crossover procedure), we perform 36 different experiments in this paper. The details of each configuration are further presented in Section 3. We also compare our method with four state-of-the-art literature methods for the generation of HMC rules, an evolutionary method based on Ant Colony Optimization (ACO) [9], and three Decision Tree induction algorithms based on the concept of Predictive Clustering Trees (PCT) [13]. Our goal here is to recommend the HMC-GA configuration that generates the best rules regarding predictive performance and interpretability.

In this paper, we focus on the Funcat taxonomy [16]. It is a tree taxonomy with up to six levels in depth and that consists of 28 main protein functional categories. This taxonomy describes functions of prokaryotic and eukaryotic proteins, like cellular transport, metabolism, and communication. A small portion of the taxonomy is illustrated in Fig. 1.

The remainder of this paper is organized as follows. In Section 2 we review some recent HMC methods to deal with PFP. Section 3 details the main aspects of HMC-GA. The methodology employed for the empirical analysis is discussed in Section 4. The experimental analysis and our findings are presented in Section 5, where 36 different HMC-GA configurations, an ACO-based method, and three decision tree variations are compared using 10 PFP datasets structured according to the Funcat taxonomy. Finally, we summarize the conclusions and suggest topics for future work in Section 6.

2. Related work

Typically, homology-based methods are used for PFP. These methods employ alignment tools to compare protein sequences against a database of proteins with known functions. This section discusses some recent HMC methods reported in the literature that make use of ML for protein and gene function prediction.

Sun et al. [17] formulated the classification task as a path selection problem, where each path starts on the root and terminates on a leaf or internal node. They used partial least squares techniques to solve the label prediction problem as an optimal path prediction

problem. Each multi-label prediction is then a connected subgraph, which can be formed by a small number of paths. The proposed method finds the optimal paths, and the final prediction is given by the union of these paths.

In Cerri [18] a method named Hierarchical Multi-label Classification with Local Multi-Layer Perceptrons (HMC-LMLP) was proposed. The method associates one multi-layer perceptron (MLP) neural network to each hierarchical level, being each MLP responsible for the predictions in its associated level. The predictions at one level are used to complement the feature vectors of the instances used to train the neural network associated to the next level. Thus, training and testing are performed in a top-down fashion.

The use of incomplete hierarchical labels was proposed by Yu et al. [12]. It takes the hierarchical and non-hierarchical similarities between protein functions and defines a combined similarity between the labels. This similarity and the known labels are used to estimate the missing functions in the hierarchy. Information regarding interactions between proteins is also used to predict their functions. A situation in which labels were missing was simulated, by randomly masking the leaf functions of a protein.

Bi and Kwok [19] use the Mandatory Leaf Node Prediction strategy (MLNP) [1]. They use hierarchy information and formulate the problem as finding the multiple labels with the largest posterior probability over all the labels. The authors extended the nested approximation property [20] to deal with HMC problems structured as DAGs, which was solved using a greedy algorithm called MAndatory leaf node prediction on Structures (MAS).

The work of Stojanova et al. [11] considers self-correlation in the HMC problem, *i.e.*, the statistical relationships between the same variable on different, but related instances. The method, called Network Hierarchical Multi-label Classification (NHMC), uses the PCT framework to build a generalized form of decision trees. During the training process, NHMC uses the instances' features and the instances' self-correlations. The method models these self-correlations as a network, and then explores these self-correlations during the learning phase.

Borges and Nievola [21] proposed a competitive neural network formed by an input layer and an output layer. The distances between the hierarchy nodes and each training instance are calculated. The neurons with the smallest distances are considered winners, influencing their ancestors. The neural network weights are adjusted according to the classes associated to the winner neurons.

The synergy between different LCN-based strategies was investigated by Cesa-Bianchi and Valentini [22]. The authors integrated Kernel-based data fusion tools and ensemble algorithms with cost-sensitive methods [10,23]. Synergy was defined as the prediction accuracy improvement, in any evaluation measure, by using concurrent learning strategies. Synergy is detected if the combined action of two strategies results in better classification rates than the average classification rates of the two strategies used separately [22].

In the work of Otero et al. [9], a global method using Ant Colony Optimization (ACO) was proposed. Rules in the format `IF ... THEN ...` are discovered, employing an ACO algorithm to optimize the rule antecedents. A sequential covering procedure creates rule antecedents that cover all (or most of the) training instances. Initially, an empty set of rules is created, and a new rule is added to the set while the number of instances not covered by any rule is larger than a given threshold.

Valentini [2] used an ensemble of LCN-based classifiers, where each classifier outputs the probability that a given instance belongs to a given class. A global consensual probability is then estimated in a combination phase. An extension was proposed in [24] and [10], in order to modulate the relationship between the prediction of a class and its class descendants.

Three Predictive Clustering Trees (PCT) methods were investigated by Vens et al. [13]: Clus-HMC, a global-based method to induce single decision tree considering all the hierarchical classes, the local-based Clus-SC, which trains a binary decision tree for each class, ignoring the relationships between the classes, and the local-based Clus-HSC, which induces a binary decision tree for each class, exploring the hierarchical relationships between them. Schietgat et al. [8] also use an ensemble technique, Clus-HMC-Ens, to combine the decision trees induced by Clus-HMC.

In Triguero and Vens [25], the authors investigated alternatives to perform the final labeling in HMC problems. The authors evaluated the Clus-HMC-Ens method when using single and multiple thresholds to transform the real-valued prediction scores into actual binary labels. To choose thresholds, two approaches were proposed: to optimize a given evaluation measure or to simulate training set properties in the test set. As evaluation measures, the authors used the hierarchical loss function and the micro-averaged f-measure, concluding that selecting thresholds for each class is a good alternative, resulting in improved label sets and faster execution time.

In Cerri et al. 2014 [14], we proposed a Genetic Algorithm to induce classification rules for protein function prediction using the Gene Ontology [26] hierarchy. That proposal was inspired by the work of Otero et al. [9], which was the first bio-inspired global-based method proposed to induce HMC rules in the literature. Before that, only Carvalho et al. 2011 [27] had used Genetic Algorithms for Hierarchical Classification, but using a local-based method, and applied to hierarchical single-label problems. Here in this paper, we still keep the main parts of the algorithm proposed by Cerri et al. 2014, such as the individual representation, operators indexation, and how to index numeric and categorical attributes. However, as pointed out in the Introduction of this paper, we work now with a different hierarchy structured as a tree and proposed new fitness functions and genetic operators in order to induce sets of rules with different characteristics. In total, we produced 36 new variations of our original method and suggests the best one to be used in the investigated datasets. Since this work is inspired in the Ant Colony Optimization-based method *hm*Ant-Miner [9], we used it as a baseline during our experimental analysis. Thus, we can compare two different nature-inspired methods proposed specifically for HMC rule induction. In addition, we compare HMC-GA with three deterministic HMC methods based on decision trees using the concept of Predictive Clustering Tress [13]. They are called Clus-HMC, Clus-HSC and Clus-SC, and are considered state-of-the-art methods for HMC rule induction.

3. Hierarchical multi-label classification with a genetic algorithm

This Section describes Hierarchical Multi-label Classification with a Genetic algorithm (HMC-GA), a global method to induce HMC rules. The main HMC-GA pseudocode is a conventional sequential instance-covering procedure to evolve the antecedents of classification rules. It is presented in Algorithm 1 [14]. In this procedure, the instances covered by a rule are removed from the training set in order to allow the generation of new rules to cover the remaining instances. The rule consequent is generated using a deterministic procedure considering the classes assigned to instances covered by the rule. We first present an overview of how HMC-GA works, and then we detail all its components, from individual representation to fitness evaluation.

Algorithm 1: A Genetic Algorithm to generate HMC rules.

```
1  HMC-GA(D, G, p, minCov, maxCov, maxNotCov, cr, mr, t, e, pt)
2  Input: training set D
3                 number of generations G
4                 population size p
5                 minimum number of instances a rule must cover minCov
6                 maximum number of instances a rule must cover maxCov
7                 maximum number of instances left uncovered maxNotCov
8                 crossover rate cr
9                 mutation rate mr
10                tournament size t
11                number of individuals selected by elitism e
12                probability of a test (attribute) to be used within a rule pt
13 Output: the set of induced rules (InducedRules)
14 inducedRules ← ∅
15 while |D|> maxNotCov do
16     initPopulation ← generateInitialPopulation(D, p, pt)
17     fitnessCalculation(initialPopulation, D)
18     currentPopulation ← initPopulation
19     bestRule ← currentPopulation best rule regarding fitness
20     j ← G
21     repeat
22         newPopulation ← ∅
23         newPopulation ← newPopulation ∪ elitism(currentPopulation, e)
24         parental ← tournament(initPopulation, t, e, p)
25         offspring ← crossover(parental, cr)
26         newPopulation ← newPopulation ∪ offspring
27         newPopulation ← mutation(newPopulation, mr, pt)
28         newPopulation ← localSearch(newPopulation, minCov, maxCov)
29         currentPopulation ← newPopulation
30         fitnessCalculation(currentPopulation, D)
31         bestRule ← recoverBestRule(initPopulation, bestRule)
32         j ← j − 1
33     until j > 0 OR ruleConvergence();
34     inducedRules ← inducedRules ∪ bestRule
35     remove from D all the instances which were covered by bestRule
36 return inducedRules
```

3.1. Overview

HMC-GA generates a classification model that comprises a set of classification rules. These rules are formed by an antecedent and a consequent. The antecedent is a vector of integer and real values, in which every four positions correspond to an operation over an attribute of the dataset. In HMC-GA, each individual is a rule antecedent. Section 3.2 explains in details how we code those antecedents.

Regarding the consequent of a rule, it is formed by an average class label vector, which is obtained from the label-space of the training instances covered by the rule. The label-space of the training instances is a matrix where each row is a binary vector of classes that were assigned to an instance, and each column is a class of the hierarchy. Thus, each instance is classified following a binary class-vector, where each position corresponds to a label. If the position is set to 1, the instance is classified as belonging to the class, with 0 meaning otherwise.

The average class vector of a given rule r is represented by a vector $\overline{\mathbf{v}}_r$ of size $|C|$, being C the set of all classes in the hierarchy. To calculate the ith component of $\overline{\mathbf{v}}_r$, we use Eq. (1). In this equation, $S_{r,i}$ is the set of all training instances covered by rule r that are classified as belonging to class i, and S_r is the set of all training instances covered by rule r. Therefore, each position $\overline{v}_{r,i}$ has the proportion of instances covered by r which are classified in class i.

$$\overline{v}_{r,i} = \frac{|S_{r,i}|}{|S_r|} \tag{1}$$

Each position of the rule consequent is a real value within the interval [0, 1]. When an instance satisfies a rule antecedent, the ith component of the consequent of that rule represents the probability the corresponding instance belongs to class i.

We implement a conventional sequential covering procedure (Section 3.5), where we generate a set of rules after a complete run of the genetic algorithm. The average class label vectors (consequents) of the rules are constructed using the labels of the instances covered by rules during the evolutionary process.

After obtaining a classification model (set of rules), we then apply the rules to a test set. To classify an unseen instance, we compare it sequentially with regard to all rules that were induced in the evolutionary process. We start with the first rule: if the instance is not covered by that rule, it is then compared with the second rule, and so on, until a rule that covers the instance is found. If the instance is not covered by any of the rules in the set, it is classified following a default rule. This default rule is simply the per-class average of the training set.

Once a given rule is found, its consequent vector is assigned to the instance at hand. Recall here that the decision making is dependent on the order of the rules, and that this order is naturally defined by the sequential covering procedure. During the evolutionary process, when a rule is found, it is stored within a list, and the instances covered by this rule are removed from the training set. The evolutionary process is repeated with the remaining instances, and the second rule found is stored as a novel rule within the list. We are aware of the ordering dependence for covering rules in the test set, and that a same test instance can be covered by more than one rule. Thus, our method does not guarantee that the best rule for a test instance is chosen. However, it guarantees that we are searching for the rules from the more general to the more specific since the first rule is generated considering the entire training set, which decreases in size as the rules are generated. We could apply many different heuristics to decide the order in which rules should be applied to the test set, but we leave a more complete investigation on the matter for future work.

Each position in the consequent vector of a rule represents the probability of a given instance to be classified as belonging to that corresponding class. Thus, a threshold has to be applied over this vector in order to provide a hard (discrete) classification. Only after obtaining this hard classification, which corresponds to a binary vector, we can evaluate the final classification assigned to the instance. The final classification of an instance as belonging to a given class is obtained by comparing each position in the class vector to a decision threshold θ. If the position value is equal to or larger than θ, the instance is classified as belonging to that class.

From Eq. (1), we can see that it handles the constraints imposed by the hierarchy of class labels. This happens because when we are building the consequent of a rule, each position of the consequent vector is filled by averaging the columns of the label-space matrix regarding the instances covered by the rule (Eq. (1)). Every time a class is predicted (value = 1), the positions corresponding to the parent classes also have to be predicted. Thus, positions corresponding to parent classes will never have values lower than positions corresponding to their children. This guarantees that when applying a threshold, we will always obtain complete hierarchical paths, respecting the hierarchy constraints.

A few studies try to find the "optimal" threshold value by modeling a threshold function as a linear function [28]. Others try to tune the threshold value by optimizing a given evaluation measure or searching for the global maximum of the evaluation measure via optimization procedures [29]. We decided for a threshold-independent procedure in order to evaluate the final classification, which is detailed in Section 4.2.

3.2. Representation of individuals

The individuals in HMC-GA are the antecedents of the classification rules. The antecedent of a rule is represented as a vector of integer and real values, in which each value is decoded into a component of the rule antecedent. The possible components are an operator, an attribute index, and an attribute value. Fig. 2 illustrates how individuals are represented as chromosomes for HMC-GA.

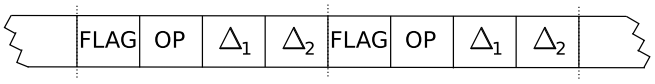


Fig. 2. Representation of individuals.
Source: Adapted from [14].

Every four positions of the chromosome vector illustrated in Fig. 2 represent a test over an attribute of the dataset. Thus, the size of the antecedent vector is four times the number of attributes and each test is coded as a 4-tuple {FLAG, OP, Δ_1, Δ_2}. The FLAG gene may be set to either 1 or 0, indicating the presence (1) or absence (0) of the corresponding test in the rule antecedent. The flag gene allows the rules to have different numbers of tests. The OP gene is an integer and must encode one of the possible operators, and genes Δ_1 and Δ_2 are real number values used as test conditions.

All tests in the rule antecedents are separated by E clauses. A given attribute A_k is tested against a value Δ using an operator OP. Different settings are used for nominal and numeric attributes. In the datasets we used in this work, all attributes are either nominal or numeric (there are no ordinal attributes). However, if we consider a dataset with ordinal attributes, they will be treated as numeric, and the operators used will be the same we use here for numeric attributes.

For nominal attributes, operators $=$, $\neq$, and *in* can be used. The *in* operator allows a given nominal attribute to be tested in order to verify if its value is among a set of values. For each nominal attribute, all its values are indexed with integer values to be used in the representation of individuals. In addition, all possible sets with two or more values formed by the nominal values of an attribute are also indexed with integers. This last procedure is necessary to allow the use of the *in* operator. When the operators $=$ and $\neq$ are used in a test, Δ_2 is set to 0 and Δ_1 is set to the index corresponding to the nominal value used in the test condition. When the operator *in* is used, Δ_2 is set to 0 and Δ_1 is set to the index corresponding to the set of nominal values used in the test condition.

For numeric attributes, operators $\leq$ and $\geq$ are used. In addition, it is possible to test a numeric attribute to check if its value belongs to a given interval ($\Delta_1 \leq A_k \leq \Delta_2$). In the single test's scenario, values Δ_1 and Δ_2 are chosen according to the test operator. If the operator $\leq$ is used, Δ_1 is set to 0 and Δ_2 receives the test value for the condition. For the operator $\geq$, Δ_2 is set to 0 and Δ_1 receives the test value for the condition. Thus, Δ_1 and Δ_2 can be seen as lower and upper bounds for numeric attributes within a test.

The procedure explained so far encodes propositional rules. To encode relational rules, we perform a modification in the previous indexation scheme. When two attributes A_1 and A_2 are compared, only operator $\leq$ is used. However, it is indexed with a different integer value, to differentiate it from the $\leq$ operator used in propositional rules. Besides, instead of real values, genes Δ_1 and Δ_2 are set to integer values representing the indexes of the attributes. Thus, when comparing attributes A_m and A_n, Δ_1 and Δ_2 are set, respectively, to integer values m and n. Recall that only numeric attributes can be compared.

3.3. Indexing of operators and nominal values

All possible operators and also all possible nominal values of an attribute are previously indexed with fixed indices. Thus, a test can be easily executed retrieving the indices and the corresponding operators and values. Fig. 3 illustrates an example of this indexing scheme.

After a rule antecedent is built, it must be capable of classifying a given instance into a set of classes, respecting the constraints of the hierarchical structure in which the classes are organized. The following example illustrates the structure of one possible HMC rule generated by HMC-GA:

IF (A_1 OP Δ) AND (A_3 OP A_5) AND (A_5 OP Δ)
THEN
$\{C_1, C_{1.1}, C_{1.2}, C_{1.3}, C_{1.2.1}, C_{1.2.2}, C_{1.3.1}\}$

3.4. Population initialization

The HMC-GA population is initialized by a seeding procedure, in which a training instance is randomly selected and its attributes are used to create an individual. After selecting an instance, each of its attributes has probability pt of being used in the antecedent corresponding to the individual being generated. Thus, there is a probability pt that the FLAG belonging to the 4-tuple corresponding to attribute A_i receives value 1.

After the FLAG of a 4-tuple receives a value (1 for active and 0 for inactive), the positions related to operators and test values are filled. The operator is randomly selected depending on whether the attribute in question is numeric/ordinal or nominal, and the position corresponding to the operator (OP) receives an index, as shown in Fig. 3.

If the operator is $=$ or $\neg$, position Δ_1 receives the index corresponding to the attribute value within the corresponding instance, and position Δ_2 receives 0. If the *in* operator is chosen, position Δ_1 receives the index corresponding to one of the sets of values that contain the attribute within the instance, and position Δ_2 receives 0. To show an example of this last procedure, if the attribute has a nominal value A, and the possible attribute values in the dataset are A, B and C, position Δ_1 receives the index corresponding to one of the possible sets of values that contain A: {A, B}, {A, C} and {A, B, C}. The set of values are thus randomly chosen. The indexing of nominal values is performed as shown in Fig. 3.

If the chosen operator works on numeric attributes, the filling of the 4-tuple values corresponding to the attribute is simpler. For the $\geq$ operator, position Δ_1 receives the attribute value, and position Δ_2 receives 0. If the $\leq$ operator is chosen, position Δ_2 receives the attribute value, and position Δ_1 receives 0. If the chosen operator corresponds to a test to verify whether the attribute value belongs to an interval ($\Delta_1 \leq A_i \leq \Delta_2$), values Δ_1 and Δ_2 are randomly chosen to make the attribute value satisfy the test condition.

If relational rules are being generated (comparing two attributes A_1 and A_2), we always use the $\leq$ operator but indexing it with a different value in order to differentiate it from the $\leq$ operator used in propositional rules. In addition, positions Δ_1 and Δ_2 now receive the attribute indices. E.g., if instance A_i and A_j are being compared, positions Δ_1 and Δ_2 receive, respectively, integer values i and j. It is important to recall that we compare only numeric attributes in relational rules.

The seeding procedure is repeated until a population with the desired size of individuals is generated. The procedure guarantees that each individual (rule antecedent) covers at least one training instance. A rule antecedent covers an instance if all its active tests are satisfied by the corresponding attributes within the instance. Fig. 4 illustrates the seeding procedure applied to an instance.

3.5. Evolutionary process

The evolutionary process starts storing the best e rules from the current population. A set of $p - e$ parent rules (being p the population size, and e the number of individuals selected by elitism) is then selected using tournament selection. Next, those parent rules are submitted to a uniform crossover procedure to generate $p - e$ child rules. In addition to the conventional uniform crossover, we also propose a specialized uniform crossover that considers the distances between average class vectors of the rules.

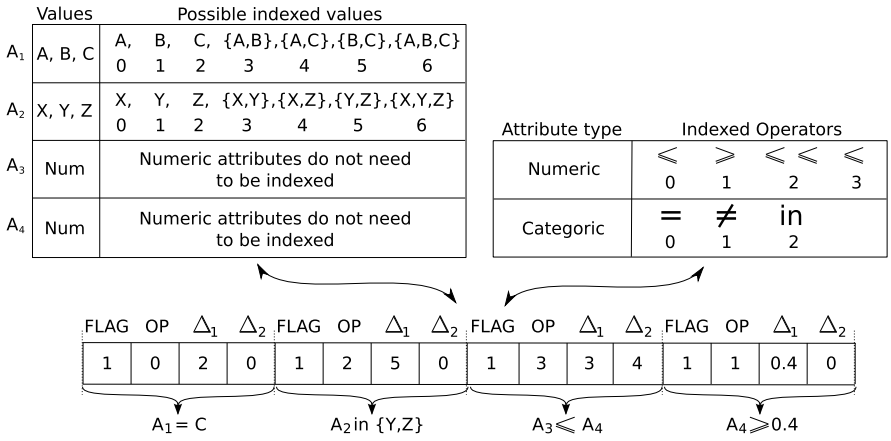


Fig. 3. Indexing of operators and nominal values.
Source: Adapted from [14].

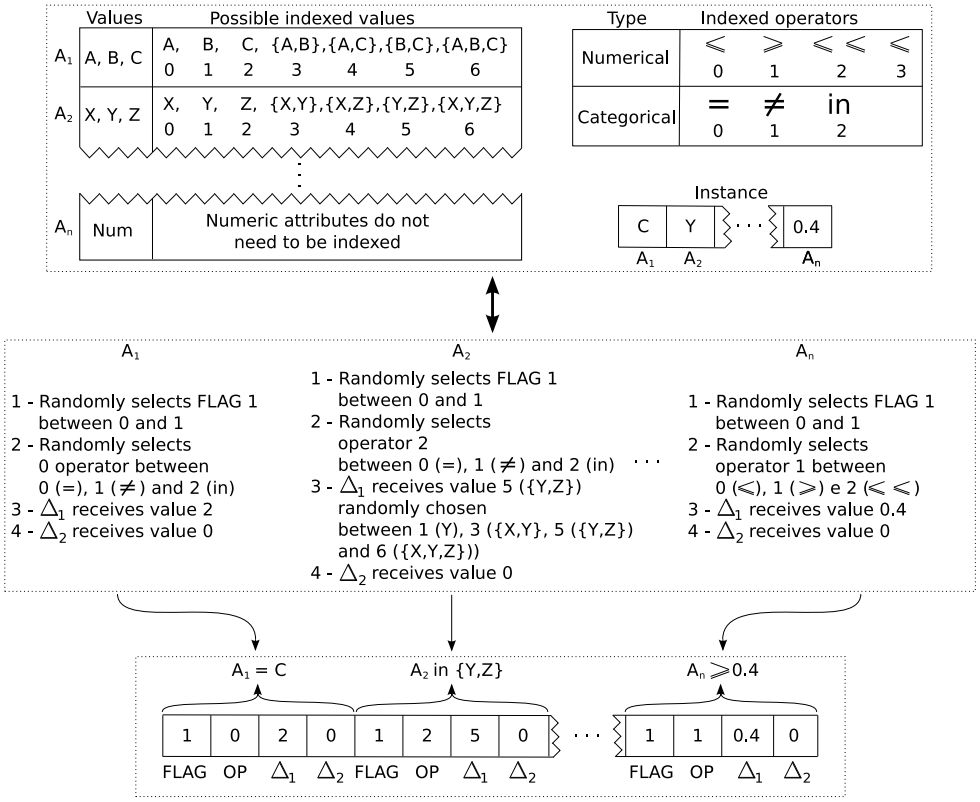


Fig. 4. Illustration of the seeding procedure.

The specialized crossover receives as input a list of $p - e$ parent rules. A rule is then removed from the list. Afterward, we calculate the Euclidean distance between the average class vector of this rule and the average class vectors of all the remaining rules in the list. The lower the distance values of two rules, the closer they are in the Euclidean space. The Euclidean distance is presented in Eq. (2). The closest rule is then removed from the list, and the two selected parent rules are submitted to uniform crossover to generate child rules. The Euclidean distance calculation is performed to apply the uniform crossover in antecedents covering close instances in the search space, i.e., instances that are classified into a similar or equal set of classes. This procedure of selecting two rules for crossover is repeated until a completely new generation is obtained.

$$euclideanDistance(\overline{\mathbf{v}}_1, \overline{\mathbf{v}}_2) = \sqrt{\sum_{i=1}^{|C|} w_i \times (\overline{v}_{1,i} - \overline{v}_{2,i})^2} \tag{2}$$

In Eq. (2), w_i corresponds to the weight associated to the ith class of the hierarchy, and $\overline{v}_{1,i}$ and $\overline{v}_{2,i}$ are the values associated to the ith position in the average class vectors $\overline{\mathbf{v}}_1$ and $\overline{\mathbf{v}}_2$. We associated weights to all classes because, in the context of hierarchical classification, similarities among classes located at levels closer to the root are more important than similarities among classes located at deeper levels [13].

The weighting scheme used is the same one used in [13]. After trying different schemes, the authors found out that the best scheme is defined according to Eq. (3). In this scheme, the weight w_0 associated to a class at the first level is defined as 0.75, and the weight of a class i is recursively defined as the multiplication of w_0 and the average weight of all its ancestors' classes P_i.

$$w_i = w_0 \times \sum_{j=1}^{P_i} w(p_j)/P_i \tag{3}$$

The reason for choosing uniform crossover is related to the positional bias present in one- or two-point crossover. Recall that the creation of new individuals by swapping genes depends on the position of the genes in the individuals' parents. In this sense, one- and two-point crossover have a strong positional bias, because the probability of two adjacent genes being swapped together is much higher than two distant genes in the individual. Based on that fact, Freitas 2002 [30] suggests the use of the uniform crossover, which presents no positional bias. Since our individual encode the antecedent of the rules, they are composed of a logical conjunction of conditions, and from a logical perspective, there is no ordering in these conditions. Of course that when encoding the rule conditions, the individuals do have a left-to-right ordering, but this is defined only for implementation purposes. Thus, the position of the attributes is arbitrary and should be ignored when encoding the antecedents of the rules. Hence the positional bias is avoided when using the uniform crossover, since pairs of attributes are swapped or not regardless of their position within the individual.

After the generation of a new set of child rules, a mutation operation is applied (Algorithm 1, line 27) to a percentage *mr* of individuals. Each one these individuals can suffer mutation in the FLAG position (to have a test included or removed from the rule with probability *pt*), or suffer a generalization/restriction operation. The generalization/restriction operations modify the Δ values of the rule's tests in order to make the rules more general or more specific. The Δ values can be increased (Eq. (4)) or decreased (Eq. (5)) according to a *factor* randomly generated within [0, 1].

$$\Delta_i = \Delta_i + (factor \times \Delta_i) \tag{4}$$

$$\Delta_i = \Delta_i - (factor \times \Delta_i) \tag{5}$$

The individuals to undergo mutation are selected according to a mutation rate *mr*. Each of the selected individuals has a 50% chance to undergo FLAG mutation or to undergo generalization/restriction, according to the operator in the test that will suffer mutation. If FLAG mutation is chosen, all 4-tuples of the individual are trespassed, and the genes corresponding to the FLAG of each test are inverted from 0 to 1, or from 1 to 0, following probability *pt*.

Note that probability *pt* is the same probability used in the seeding procedure for the population initialization. Usually, low *pt* values are used in the seeding procedure, because high values would result in the creation of rules with many active tests, thus covering very few instances or only those used as seed. This same logic is adopted for the mutation within the FLAG genes, because high *pt* values would activate many tests, resulting in a rule covering very few or no examples whatsoever.

When the FLAG mutation is not chosen, all active tests are traversed in order to perform the generalization/restriction operation. If the test attribute is numeric, a random choice is made between generalization and restriction. If generalization is chosen, it is applied according to the operator presented in the test being generalized, choosing adequate values to be added to or subtracted from the gene values corresponding to Δ_1 and Δ_2. The same occurs if the restriction operator is chosen.

If the generalization/restriction mutations are applied to a nominal attribute, their application depends on the operator within the test. If this operator is $=$, the test can only be generalized. This generalization occurs by replacing $=$ by operators $\neq$ or *in*. In this case, one of those two operators is randomly chosen. If the operator $\neq$ is selected, it simply replaces the $=$ operator in the test. If the operator *in* is chosen, an additional procedure is necessary: the current nominal value in the test condition should be replaced by a set of values that contain the current value. If more than one set is possible, one of them is randomly selected.

For the $\neq$ operator, the test can be restricted if it is replaced by $=$. Still, we can replace the $\neq$ operator by *in*. In that last case, the test can be generalized or restricted depending on the values selected to the test. As an example, suppose that the test condition is $A_i \neq$ B, and that the possible values for A_i are A, B, C and D. If the test condition is replaced by A_i *in* {A, C}, the test is restricted. However, if the test condition is replaced by A_i *in* {A, B, C, D}, the test is generalized.

If *in* is the test operator, the test can also be generalized or restricted, depending on the new values selected for the test. The current set of values can be replaced by a set with fewer or more values, and also by a single value. If the current set of values is replaced by a single value, the *in* operator is replaced by $=$.

After the application of the mutation operators and generation of a new set of individuals, a local search operator is then executed (Algorithm 1, line 28). This operator tries to guarantee that the generated rule antecedents cover between a minimum and a maximum number of instances, making the rules neither too specific nor too general. The minimum number (*minCov*) and the maximum number (*maxCov*) covered by a rule are user-specified parameters. The local search operator is presented in Algorithm 2.

Algorithm 2: HMC-GA local search operator.

```
1  procedure localSearch(population, minCov, maxCov)
2  Input: population of individuals population
3                minimum number of instances covered by a rule minCov
4                maximum number of instances covered by a rule maxCov
5  Output: modified population newPopulation
6  foreach individual from population do
7      maxTries ← 0
8      convergency ← 0
9      while convergency = 0 AND maxTries < MAX do
10         maxTries ← maxTries + 1
11         if number of covered instances < minCov then
12             case number of active tests > 1
13                 individual ← removeActiveTest(individual)
14             case number of active tests = 1
15                 replace test from individual by a newTest randomly chosen
16                 if numeric attribute then
17                     newTest ← generalizationNumeric(test)
18                 else
19                     newTest ← generalizationCategoric(test)
20                 individual ← updateTest(newTest)
21         else if number of covered instances > maxCov then
22             if number of active tests < number of dataset attributes then
23                 individual ← addAtiveTest(individual)
24         else
25             convergency ← 1
26 newPopulation ← population
27 return newPopulation
```

Some possible situations are worth mentioning regarding Algorithm 2. The first situation relates to the maximum number of attempts executed for a rule convergence. After a *MAX* number of attempts, if both criteria *minCov* and *maxCov* are not satisfied, the final rule antecedent is the one obtained in the last attempt. This stop criterion is necessary because a rule may never satisfy the given criteria.

The second situation is related to the procedure executed when a rule is too specific (Algorithm 2, line 11). If the number of active tests in the antecedent of the rule is higher than 1, one test can be randomly removed in order to make the rule more general. However, it may occur that a rule has only one active test, and still be too specific. In that case, the test in question is replaced by another randomly chosen test, which is generalized depending on whether the attribute in the test is numeric or nominal.

A third situation is observed when a rule is too general (Algorithm 2, line 21). In this case, a test should be added to the rule antecedent, in order to make it more specific. However, it may occur that a rule has all its tests active (uses all dataset attributes) and still be too general. In that case, no procedure is performed because such a rule is very rare, and may contribute positively to the generation of future rules.

At the end of the generation of a new population, the best rule is stored (Algorithm 1, line 31). For the selection of the best rule, two factors are considered: the rule fitness, and whether it covers the minimum number of instances specified by *minCov*. Initially, when the first generation of individuals is created, the best rule is stored according to its fitness. From the second generation onwards, all rules are ordered according to their fitness values and compared to the best rule of the previous generation. If a new rule has its fitness value higher than the fitness value of the best rule from the previous generation, and also covers a minimum number of instances *minCov*, it will become the best current rule. The evolutionary process (Algorithm 1, from lines 21 to 32) is executed until a maximum number of generations is reached, or until the population converges. The population converges when the same rule remains the best after a specific number of generations.

At the end of the evolutionary process, the best rule is stored in a set of rules (Algorithm 1, line 34), and the instances covered by this rule are removed from the training set (Algorithm 1, line 35). The evolutionary process is then restarted with the creation of a new set of initial rules. This sequential covering procedure is repeated until all, or most (*maxNotCov* parameter from Algorithm 1) training instances are covered. If there are still uncovered instances in the training set, they are classified using a default rule. The default rule simply classifies the instances using the average class label vector considering all classes in the training set.

3.6. Fitness calculation

Once the consequents of the rules are built, they are evaluated by the fitness function. We used three fitness functions to evaluate the rules: (i) variance gain, (ii) weighting of variance gain and percentage of covered instances, and (iii) weighting of variance gain and area under the precision–recall curve. These three fitness functions are detailed next.

3.6.1. Variance gain

This fitness function is based on the variance gain ($Fitness_{varGain}$) [9,13] and is presented in Eq. (6).

$$Fitness_{varGain}(r, S) = var(S) - \frac{|S_r|}{|S|} \times var(S_r) - \frac{|S_{\neg r}|}{|S|} \times var(S_{\neg r}) \quad (6)$$

According to Eq. (6), the training set S is divided into two subsets: the set of instances covered by rule r (S_r), and the set of instances not covered by rule r ($S_{\neg r}$). The variance gain of rule r is then calculated with respect to the set S. The calculation also involves the variance (var) of the set of instances covered and not covered by rule r. This variance is defined as the mean square distance between the class vector of each instance, and the average class label vector of the set of instances being considered (S, S_r or $S_{\neg r}$). The variance calculation for a set of instances is presented in Eq. (7). The distance used is the weighted Euclidean distance, previously presented in Eq. (2).

$$var(S) = \frac{\sum_{k=1}^{|S|} euclideanDistance(\mathbf{v}_k, \overline{\mathbf{v}})^2}{|S|} \quad (7)$$

3.6.2. Weighting of variance gain and percentage of covered instances

This second fitness function combines the variance gain with the percentage of instances covered by the rule. The percentage of instances covered is given by Eq. (8). Recall that the size of the training set S decreases as the sequential covering procedure is executed. Previously-covered instances are removed from the training set.

$$percentCover(r) = \frac{|S_r|}{|S|} \quad (8)$$

The fitness function weighting the variance gain and percentage of covered instances is given by Eq. (9).

$$Fitness_{varGain,Cover}(r, S) = 2 \times \frac{Fitness_{varGain}(r, S) \times percentCover(r)}{Fitness_{varGain}(r, S) + percentCover(r)} \quad (9)$$

3.6.3. Weighting of variance gain and area under the precision–recall curve

We also calculated a fitness function based on the Area Under the Precision–Recall Curve ($AU(\overline{PRC})$) and the variance gain. The $AU(\overline{PRC})$ is the evaluation measure used in our experiments and is detailed in Section 4. Eq. (10) shows this fitness calculation.

$$Fitness_{varGain,AU(\overline{PRC})}(r, S) = 2 \times \frac{Fitness_{varGain}(r, S) \times AU(\overline{PRC})}{Fitness_{varGain}(r, S) + AU(\overline{PRC})} \quad (10)$$

3.7. Computational complexity

The complexity analysis for GAs is complex and often not performed at all [31]. Nevertheless, we present here the computational complexity of the most critical procedures within HMC-GA. They are the seeding procedure used to initialize the population, the local search operator, and the fitness calculation.

3.7.1. Population initialization

The population initialization is considered a critical operation because all attributes of a seed instance must be visited. Still, for each attribute, four positions are considered in the vector representing the individual being constructed. Thus, given $|A|$ the number of attributes, and *pop* the number of individuals in the population, the computation complexity of the seeding procedure is given by $\mathcal{O}(|A| \times 4 \times pop) = \mathcal{O}(|A| \times pop)$.

3.7.2. Local search

The local search procedure is executed while a rule does not converge. There are two scenarios here for the worst case. In the first one, all attributes are used in a rule, which makes it too specific, covering very few instances. In this case, the number of active terms is equal to $|A|$ (number of attributes). The worst case occurs when $|A|-1$ attributes in the rule antecedent should be deactivated ($\texttt{FLAG} = 0$), in order to make the rule cover a desirable number of instances. Thus, all attributes need to be visited, resulting in computational complexity of $\mathcal{O}(|A|)$. The second scenario occurs when the rule has only one active attribute, being too general, and all its attributes must be activated ($\texttt{FLAG} = 1$) in order to make the rule cover a desirable number of instances. In this case, all attributes should also be visited, resulting in a computational complexity of $\mathcal{O}(|A|)$.

In addition to visiting all attributes of a rule, every time an attribute is activated or deactivated, the number of covered instances should be computed. For this operation, each active attribute should be compared to the corresponding attribute value in all instances. Consider the worst-case scenario, in which a rule has only one active attribute, and needs all its attributes to be activated in order to cover a desirable number of instances. Thus, after activation of the second attribute, two attributes of the rule should be compared to their corresponding attribute values in the instances. With $|X|$ the total number of instances, $|X| \times 2$ operations are performed. Because each comparison involves a $4-tuple$ associated to the attribute, a total of $|X| \times 2$ comparisons is performed. Eq. (11) gives the cost of the comparisons as the number of active attributes increases. The equation is quadratic in the number of attributes $|A|$. Thus, the computational complexity of the local search is given by $\mathcal{O}(|X| \times |A|^2)$.

$$\begin{aligned} Cost_{comp} &= 2 \times 4 \times |X| + 3 \times 4 \times |X| + \cdots + |A| \times 4 \times |X| \\ &= |X| \times 4 \times \sum_{i=2}^{|A|} i = |X| \times 4 \times \left(\frac{|A|(|A|+1)}{2} - 1 \right) \end{aligned} \quad (11)$$

3.7.3. Fitness functions

To calculate the fitness functions used in the experiments, the most critical procedure is the variance calculation of a given rule. We need to calculate the variance of all training instances, and also the variance of the sets of covered and uncovered instances of the rule. The variance of a set of instances first calculates the average class label vector (prototype) of the instances. Afterward, it is necessary to calculate the Euclidean distances between the label vectors of all instances and the prototype.

The average class label vector of a set of instances is given by the sum of all v_j values from the label vectors $\mathbf{v}$ of each instance, divided by the total number of instances in the set. Thus, being $|C|$ the number of problem classes, the number of necessary operations to calculate the prototype is given by $|X| \times |C|$. Provided that S_r and $S_{\neg r}$ are the set of covered and uncovered instances of a rule r, the number of operations needed to construct the mean class label vectors of these two groups is given by $|S_r| \times |C|$ and $|S_{\neg r}| \times |C|$.

After obtaining the average class label vectors, the Euclidean distances between each instances' label vectors from each set of instances ($X, S_r eS_{\neg r}$) and the corresponding prototype vectors of these sets are calculated. For such, $|C|$ operations are required. Thus, the Euclidean distance calculation involving all instances from all sets of instances requires, respectively, $|X| \times |C|$, $|S_r| \times |C|$ and $|S_{\neg r}| \times |C|$ operations. Thus, the computational complexities associated to the calculations of the variances of the sets X, S_r and $S_{\neg r}$ are, respectively, $\mathcal{O}(|X| \times |C|)$, $\mathcal{O}(|S_r| \times |C|)$ and $\mathcal{O}(|S_{\neg r}| \times |C|)$.

4. Methodology

This Section presents the genetic algorithm variations, and the corresponding parameter values, the datasets, and the evaluation measures used in the experiments. The section also briefly presents *hm*Ant-Miner and the decision tree induction algorithms used as baselines in the experimental evaluations.

4.1. Datasets

Ten protein function prediction datasets are used in the experiments. Their features are related to issues like phenotype and gene expression levels. The hierarchy follows the Funcat taxonomy, and is organized as a tree. The data is freely available at the KU Leuven Declarative Languages and Artificial Intelligence Group repository.[1]

[1] https://dtai.cs.kuleuven.be/clus/hmcdatasets/.

Vens et al. [13] divided these datasets in training, validation and testing subsets. We are using these same partitions. We joined the training and validation partitions to form a complete training set, and evaluated the generated rules in the testing set. Table 1 [4] presents their main characteristics, regarding the number of classes and instances. We present a brief description of each dataset. Further details can be found in Vens et al. [13] and in the corresponding references.

- **1 - Seq**: features representing statistics obtained directly from the sequences, such as amino acid rates, sequence length and molecular weight. Most of the values are real numbers obtained using the ProtParam software [32] or taken from the MIPS repository [33];
- **2 - Pheno**: features related to phenotypical data, representing knock-out mutants missing, regarding their growth or lack of growth. The data were collected from databases that include MIPS [33] and TRIPLES [34]. The dataset is sparse, and has discrete feature values;
- **3 to 10**: microarray features used to test the expression levels of genes across genomes. They have real value attributes [13].

4.2. Evaluation measures

As discussed in Section 3, for each class, HMC-GA outputs real values in the interval [0,1]. Thus, a threshold value was used to obtain the final predictions. To classify an instance into a given class, if the output value corresponding to the class is equal to or larger than the threshold, the instance is assigned to the class.

The choice of an "optimal" value for the threshold is a difficult task, since low values lead to many predictions for each instance, resulting in high recall and low precision. Large values lead to very few predictions, resulting in high precision and low recall values. We dealt with this problem using precision–recall curves (PR-curves) [43]. A PR-curve is produced applying threshold values in the interval [0, 1] to the outputs of the classifiers. This results in different precision and recall values (points within the PR space), one for each threshold used. The union of these points forms a PR-curve, and the area under the curve can be calculated. The areas under the PR-curves can be used to compare different methods.

To calculate the area under the PR-curve, an interpolation of the precision–recall points (PR-points) [43], and posterior connection, is required. Connecting the points without interpolation would artificially increase the area below the curve. To calculate the PR points ($\overline{Prec}$, $\overline{Rec}$), we used Eqs. (12) and (13). These equations are the micro-average of precision and recall. The index i, in the equations, ranges from 1 to $|C|$. The number of true positives, false positives, and false negatives, are represented, respectively, by TP, FP, and FN.

$$\overline{Prec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i} \quad (12)$$

$$\overline{Rec} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i} \quad (13)$$

With the precision and recall points, we calculate the area under the average PR-curve ($AU(\overline{PRC})$) [13]. Its value is in the interval [0, 1], where the higher, the better.

The statistical significance of the results was assessed using the non-parametric Friedman and Nemenyi statistical tests, suitable when comparing many classifiers using several datasets [44]. The confidence level of 95% was adopted. We used exactly the same partition provided by Vens et al. 2008 [13], 2/3 of each dataset were used to train the classifiers and 1/3 for testing.

Table 1
Summary of datasets: Number of attributes ($|A|$), number of classes ($|C|$), number of classes per level (Classes per level), total number of instances (Total) and number of multi-label instances (Multi).

Dataset	$\|A\|$	$\|C\|$	Classes per level	Training		Valid		Test	
				Total	Multi	Total	Multi	Total	Multi
1 - Seq [35]	478	499	18/80/178/142/77/4	1701	1344	879	679	1339	1079
2 - Pheno [35]	69	455	18/74/165/129/65/4	656	537	353	283	582	480
3 - Cellcycle [36]	77	499	18/80/178/142/77/4	1628	1323	848	673	1281	1059
4 - Church [37]	27	499	18/80/178/142/77/4	1630	1322	844	670	1281	1057
5 - Derisi [38]	63	499	18/80/178/142/77/4	1608	1309	842	671	1275	1055
6 - Eisen [39]	79	461	18/76/165/131/67/4	1058	900	529	441	837	719
7 - Expr [35]	551	499	18/80/178/142/77/4	1639	1328	849	674	1291	1064
8 - Gasch1 [40]	173	499	18/80/178/142/77/4	1634	1325	846	672	1284	1059
9 - Gasch2 [41]	52	499	18/80/178/142/77/4	1639	1328	849	674	1291	1064
10 - Spo [42]	80	499	18/80/178/142/77/4	1600	1301	837	666	1266	1047

4.3. Genetic algorithm variations

This section lists the variations performed in the proposed HMC-GA. The variations resulted in 36 different configurations. Thus, we ended up with 36 experiments for each dataset.

4.3.1. Variation in the fitness function

We considered the three previously presented fitness function variations in our experiments. For simplification purposes, we refer to them as F1, F2 and F3.

- F1 — Variance Gain;
- F2 — Ponderation between variance gain and percentage of covered instances;
- F3 — Ponderation between variance gain and $AU(\overline{PRC})$.

The ponderation in the fitness function was the harmonic mean between the terms involved, similar to the traditional F_1 measure.

4.3.2. Variation in the crossover procedure

We also considered four variations in the HMC-GA evolutionary process. These variations were implemented through modifications in the crossover operator, and also through the use of a local search operator. We refer to these variations as C1, C2, C3 and C4.

- C1 — Uniform crossover without local search;
- C2 — Uniform crossover with local search;
- C3 — Distance-based Uniform crossover without local search;
- C4 — Distance-based Uniform crossover with local search.

4.3.3. Variation in the constructed rules

Finally, we considered the induction of three different types of rules, which we refer as R1, R2 and R3.

- R1 — Only rules with propositional tests. These rules evaluate if an attribute value A_k satisfies a test condition, *e.g.* $A_k \leq x_{i,k}$;
- R2 — Only rules with relational tests. These rules have tests only comparing the values of different attributes, *e.g.* $A_1 \leq A_2$;
- R3 — Rules with both propositional and relation tests, randomly mixed.

4.4. Genetic algorithm parameter values

Table 2 presents the parameter values used in the experiments with HMC-GA. The *pt* parameter refers to the probability of using an attribute when creating the antecedents of the initial population. A lower probability generates smaller antecedents, with higher chances of covering instances. This same probability is also used when performing the mutation in the FLAGS of the antecedents. During this mutation, the use of a high probability value would activate too many tests in the antecedent, resulting in a rule covering very few or none instances.

Still regarding the *pt* parameter value, Table 2 shows that it is given by the expression $|A| \times pt = 5$, in which A is the set of attributes in the dataset. Thus, the choice of the *pt* probability is performed in order to obtain antecedents with an average of five attributes.

The *maxNotCov* parameter refers to the maximum number of instances that can be left uncovered by any rule. If, at the end of the evolutionary process, the number of remaining uncovered instances in the training set is lower or equal to *maxNotCov*, the sequential covering process is stopped. The uncovered instances are classified in the default rule.

The *minCov* parameter defines the lower bound for the number of covered instances for each rule. It is used to avoid the creation of too specific rules. This parameter, together with the *maxCov* parameter, is also used in the local search procedure. The objective is to construct neither too specific nor too general rules.

The *G* parameter refers to the maximum number of generations evolved to obtain a classification rule. After this maximum number, the best rule is saved, and its covered instances are removed from the training set. The evolutionary procedure is then restarted with new initial rules.

The parameter values presented in Table 2 were based on the work proposed by Carvalho et al. [27], which developed a Local Classifier per Node-based method using genetic algorithms. To our knowledge, it is the only method which uses genetic algorithms for hierarchical classification. However, the method was proposed only for single-label hierarchies.

4.5. The baseline hmAnt-Miner method

The *hm*Ant-Miner method [9] is a global-based method that uses Ant Colony Optimization (ACO) to evolve antecedents of HMC rules. Rules in the format IF ... THEN ... are discovered, with the ACO algorithm used to optimize the rule antecedents. A sequential covering procedure creates rule antecedents that cover all (or most of the) training instances. Initially, an empty set of rules is created, and a new rule is added to the set while the number of instances not covered by any rule is larger than a given threshold.

Similar to HMC-GA, the consequent of a rule is generated by a deterministic procedure, based on the set of instances covered by the rule. The heuristic used to evaluate the rules is also the variance gain, based on the Euclidean distance between the rule consequents.

Table 3 presents the parameter values used by *hm*Ant-Miner in the experiments. These values are the same used in Otero et al. [9].

4.6. The baseline PCT-based methods

The Predictive Clustering Trees (PCTs) [45] framework constructs decision trees as a cluster hierarchy. It begins with a root node containing all the training instances, and then recursively

Table 2
HMC-GA parameter values.

Parameter	Description	Value
p	Population size	50
e	Number of individuals selected by elitism	1
mr	Mutation rate	40%
cr	Crossover rate	90%
pt	Probability of using a test in a rule	$\|A\| \times pt = 5$
G	Number of generations	50
t	Tournament size	2
maxNotCov	Maximum number of uncovered instances	10
minCov	Minimum number of covered instances per rule	5
maxCov	Maximum number of covered instances per rule	300

Table 3
*hm*Ant-Miner parameter values.

Parameter	Description	Value
maxUncoveredInstances	Maximum number of uncovered instances	10
maxNumberIterations	Maximum number of iterations	1500
ruleConvergence	Number of iterations used to test the rule convergence	10
minInstancesPerRule	Minimum number of covered instances per rule	10
colonySize	Number of ants per iteration	30

partitions it in small clusters as the decision tree is traversed towards the leaves. The framework can be applied both to clustering and classification problems, and the decision trees are induced using top-down strategies similar to others used in the literature, such as C4.5.

We used three PCT-based methods in our experiments, presented by Vens et al. [13]: Clus-HMC, a global-based method to induce a single decision tree considering all the hierarchical classes, the local-based Clus-SC, which trains a binary decision tree for each class, ignoring the relationships between the classes, and the local-based Clus-HSC, which induces a binary decision tree for each class, exploring the hierarchical relationships between them.

Similar to HMC-GA and *hm*Ant-Miner, the PCT-based methods also use the variance gain to decide which attribute gives the best split in the data. The variance of a set of instances is given by the mean square Euclidean distance between each instances' class label vectors $\mathbf{v}_i$ and the mean class label vector $\bar{\mathbf{v}}$ of the instances in the set.

Table 4 presents the parameter values used by the PCT-based methods in the experiments. These values are the same used in Vens et al. [13]. For the F Test parameter, all values in the table were used in the validation dataset in order to choose the best one. The chosen one was then used to induce a decision tree using the training and validation datasets together.

As explained in Sections 4.5 and 4.6, all baseline methods in the literature adopt the Euclidean distance within their implementation. The *hm*Ant-Miner uses the Euclidean distance when calculating its fitness function. It also uses Variation Gain, calculating the Euclidean distances between binary vectors representing the classes in the label-space. The same strategy is also adopted by the three Predictive Clustering Trees (PCT) baseline methods, which use the Variation Gain to decide which attribute gives the best split in the data. They used the Euclidean distance also considering the binary class-label vectors. This is why we opted to use the Euclidean distance within our Genetic Algorithm. We consider it a good choice to use the Euclidean distance in our method in order to have a fair comparison regarding Variation Gain. Of course many other distances could be used, but this would require modifications in the baseline methods. We believe this is out of the scope of this contribution, but we sure consider it for future works. In addition, in the work of Aleksovski et al. [46], the authors performed a study comparing Euclidean distance, Jaccard distance, SimGIC distance and ImageCLEF distance for HMC problems using the PCT-based methods. They concluded there were no statistically significant differences among the different distance measures.

4.7. Statistical analysis

To evaluate the statistical significance of the experimental results, we present the results of statistical tests by following the approach proposed by Demšar [44]. In brief, this approach seeks to compare multiple algorithms on multiple datasets, and it is based on the use of the Friedman test with a corresponding post-hoc test. The Friedman test is a non-parametric counterpart of ANOVA, as follows. Let R_i^j be the rank of the jth of k algorithms on the ith of N datasets. The Friedman test compares the average ranks of the algorithms, $R_j = \frac{1}{N}\sum_i R_i^j$. The Friedman statistic (Eq. (14)) is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are large enough.

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{14}$$

Iman and Davenport [47] showed that Friedman's χ_F^2 is undesirably conservative and derived an adjusted statistic, given by Eq. (15), which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

$$F_f = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2} \tag{15}$$

If the null hypothesis of similar performances is rejected, we proceed with the Nemenyi post-hoc test for pairwise comparisons. The performance of two classifiers is significantly different if their corresponding average ranks differ by at least a critical difference, given by Eq. (16), where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}} \tag{16}$$

5. Experiments and discussion

This Section presents and analyzes the main experimental results. In total, 36 experiments were performed. Table 5 lists all combinations adopted for each experiment.

The radar plots from Figs. 5 to 9 show the comparative results of all the 36 experiments in the ten datasets investigated. For each experiment, we executed HMC-GA ten times, each time randomly generating initial rules. Thus, the values shown are the average $AU(\overline{PRC})$ values over all executions. Recall that the showed results

Table 4
PCT-based methods parameter values.

Parameter	Description	Value
WType	Weight of a class, being w_0 times the average of the parent's class weights	Average
MinimalWeight	Minimal number of instances in each cluster	5
FTest	Stopping criterion: A node will only be split if a statistical F-test indicates a significant reduction of variance inside the subsets	0.001, 0.005, 0.01, 0.05, 0.1, 0.125

Table 5
List with all combinations performed for each experiment.

Exp. 1 (E1) - F1 x C1 x R1	Exp. 2 (E2) - F1 x C1 x R2	Exp. 3 (E3) - F1 x C1 x R3
Exp. 4 (E4) - F1 x C2 x R1	Exp. 5 (E5) - F1 x C2 x R2	Exp. 6 (E6) - F1 x C2 x R3
Exp. 7 (E7) - F1 x C3 x R1	Exp. 8 (E8) - F1 x C3 x R2	Exp. 9 (E9) - F1 x C3 x R3
Exp. 10 (E10) - F1 x C4 x R1	Exp. 11 (E11) - F1 x C4 x R2	Exp. 12 (E12) - F1 x C4 x R3
Exp. 13 (E13) - F2 x C1 x R1	Exp. 14 (E14) - F2 x C1 x R2	Exp. 15 (E15) - F2 x C1 x R3
Exp. 16 (E16) - F2 x C2 x R1	Exp. 17 (E17) - F2 x C2 x R2	Exp. 18 (E18) - F2 x C2 x R3
Exp. 19 (E19) - F2 x C3 x R1	Exp. 20 (E20) - F2 x C3 x R2	Exp. 21 (E21) - F2 x C3 x R3
Exp. 22 (E22) - F2 x C4 x R1	Exp. 23 (E23) - F2 x C4 x R2	Exp. 24 (E24) - F2 x C4 x R3
Exp. 25 (E25) - F3 x C1 x R1	Exp. 26 (E26) - F3 x C1 x R2	Exp. 27 (E27) - F3 x C1 x R3
Exp. 28 (E28) - F3 x C2 x R1	Exp. 29 (E29) - F3 x C2 x R2	Exp. 30 (E30) - F3 x C2 x R3
Exp. 31 (E31) - F3 x C3 x R1	Exp. 32 (E32) - F3 x C3 x R2	Exp. 33 (E33) - F3 x C3 x R3
Exp. 34 (E34) - F3 x C4 x R1	Exp. 35 (E35) - F3 x C4 x R2	Exp. 36 (E36) - F3 x C4 x R3

were obtained in the test partition. To induce the rules, we joined the training and validation partitions, forming a unique training set.

Looking at the radar plots, it is possible to identify some patterns within the experiments. The bottom part of each chart concentrates all experiments where the fitness function used was the ponderation between variance gain and percentage of covered instances (F2). We can see that, in general, this fitness function led to the best $AU(\overline{PRC})$ results.

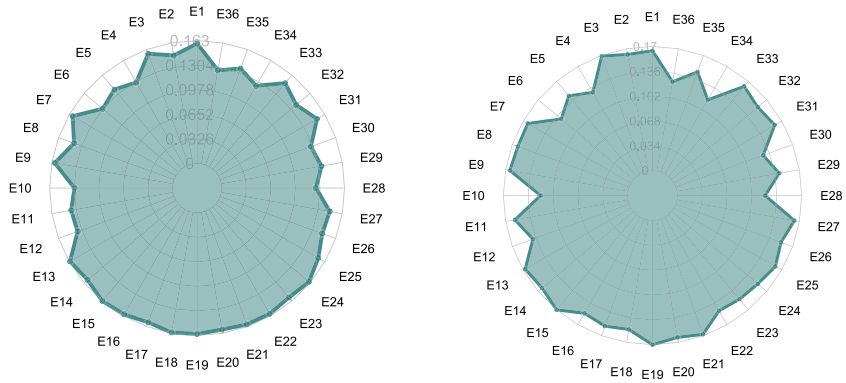
The radar plots also give insights regarding the experimental configurations that led to the worst results regarding the $AU(\overline{PRC})$ values. The charts show that, in general, poor results were obtained by experiments E4, E5 and E6. These three experiments have fixed configuration values for the fitness function, using the Variance Gain (F1), and for the crossover operation, using a uniform crossover with local search (C2). The experiments E10, E11, and E12 presented, in general, $AU(\overline{PRC})$ values among the worst obtained. They all also used Variance Gain (F1) as fitness function and distance-based uniform crossover with local search (C4). Thus, the experiments suggest that our local search combined with only variance gain as fitness function is not a good experimental configuration. In all the above configurations, the three types of rules (R1, R2, and R3) were generated and seemed not to influence the results.

The experimental configurations E28, E29, and E30 also presented, in general, poor results. Their fitness function was the ponderation between variance gain and $AU(\overline{PRC})$ (F3), and they used the uniform crossover with local search (C2). The same fitness function (F3) was used in the experiments E34, E35 and E36, combined with distance-based uniform crossover with local search (C4). Thus, again the results suggest that local search combined with variance gain is not a good experimental configuration. Additionally, the variation in the types of rules generated (R1, R2, and R3), considering the above configurations, seemed not to directly influence the results.

Given the difficulty in analyzing many experimental results in different datasets individually, we also analyzed the average performances of the experiments considering all datasets. For such, Figs. 10–12 present boxplots considering our proposed variations and also the baseline methods. The *hm*Ant-Miner method is shown in the graphs under the name ACO. As we did for HMC-GA, we also executed *hm*Ant-Miner ten times, and averaged the obtained $AU(\overline{PRC})$ values, the number and the size of the rules. The PCT-based methods Clus-HMC, Clus-HSC and Clus-SC are also shown in the graphs, respectively as HMC, HSC and SC. Because they are deterministic methods, only one execution is necessary. We did not show the methods Clus-HSC and Clus-SC in the graphs regarding rules and tests because these methods produced a huge number of rules, varying from nearly 2000 and 10000 rules. Thus, as they clearly do not generate interpretable models, we left them out of this analysis. We sorted the boxplots according to the ranking obtained by the Friedman test.

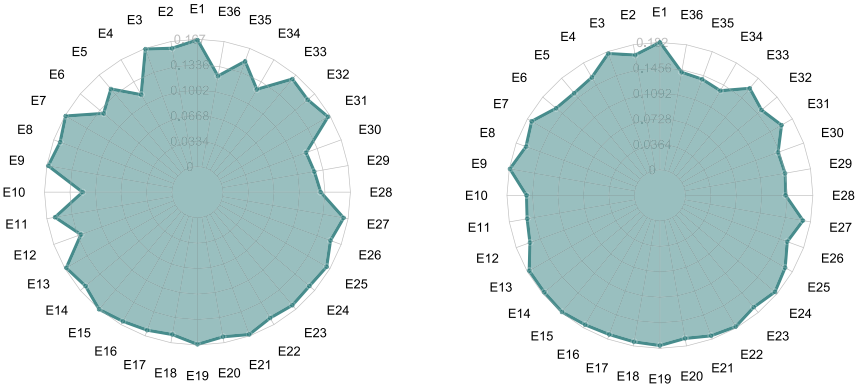
The results of Fig. 10 confirm what is observed in the graphs from Figs. 5 to 9. Considering the genetic algorithm variations, the experiments E15 (F2 $\times$ C1 $\times$ R3), E13 (F2 $\times$ C1 $\times$ R1), and E19 (F2 $\times$ C3 $\times$ R1) are among the top best average results. This shows that our fitness function using ponderation between variance gain and percentage of covered instances can be a good choice. Besides, better results were obtained without the use of the local search operator and using propositional tests in the rules (combined or not with relational tests). The number of instances covered by a rule also influences its performance, since rules covering a huge number of instances are too general, while rules covering a very small number of instances are too specific. Thus, we considered this aspect in the fitness function, which seemed to improve the quality of the rules in terms of interpretability and performance. It is important to recall that our fitness function equally weights variance gain and percentage of covered instances.

Among the top best $AU(\overline{PRC})$ values, we can also find the HMC-GA configurations from the experiments E1 (F1 $\times$ C1 $\times$ R1), E3 (F1 $\times$ C1 $\times$ R3), and E7 (F1 $\times$ C3 $\times$ R1). According to these results, our local search (configurations C1 and C3) should not be used. Again, rules containing propositional tests led to better results. Our hypothesis is that relational tests largely increase the search space when comparing different attributes among themselves. While this potentially increases the possibility of finding good solutions, this also can make the classification problem more difficult, since good solutions are more difficult to find in larger search spaces [14].



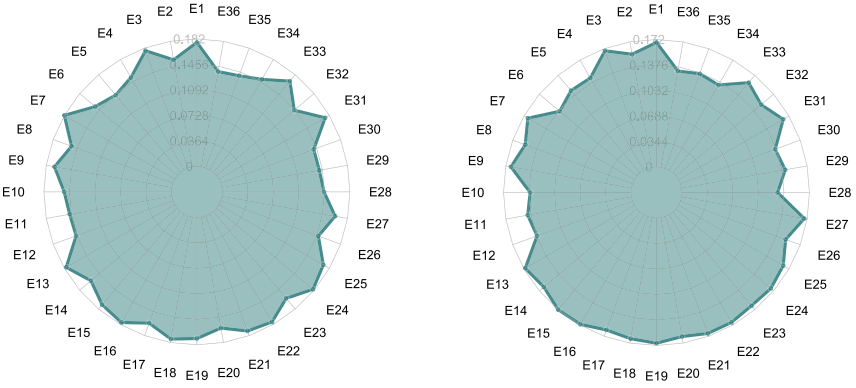
(a) Cellcycle (b) Church

Fig. 5. Results for the Cellcycle and Church datasets.



(a) Derisi (b) Eisen

Fig. 6. Results for the Derisi and Eisen datasets.

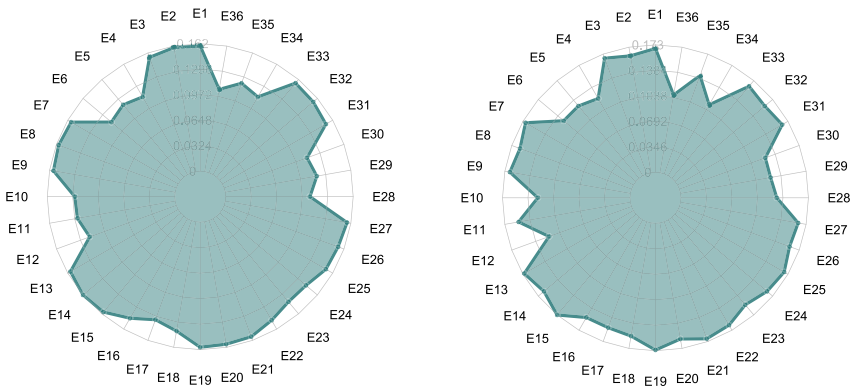


(a) Gasch1 (b) Gasch2

Fig. 7. Results for the Gasch1 and Gasch2 datasets.

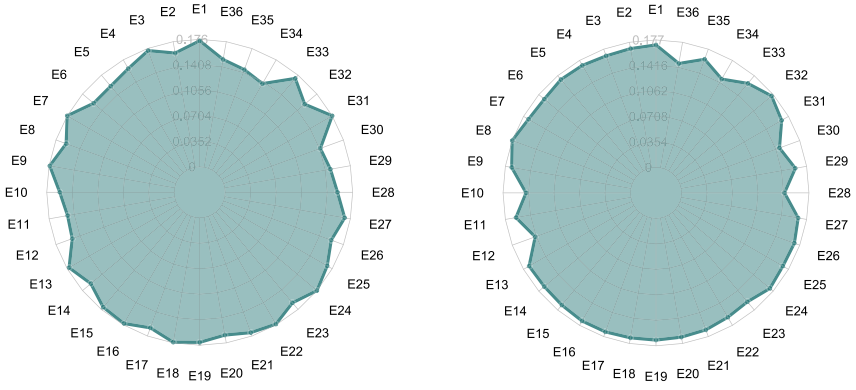
HMC-GA with the configuration of Experiment 7 (E7) is closely followed by *hm*Ant-Miner in terms of $AU(\overline{PRC})$. Recall that *hm*Ant-Miner also uses only variance gain to evaluate its induced rules and also induces rules with only propositional tests.

We also have to consider that the variance gain used in the fitness functions has a characteristic that may harm the performances of the methods in some situations. According to Eq. (6), maximizing the variance gain of a rule corresponds to minimizing the difference between the variances of S_r and $S_{\neg r}$. However, in the case where a very homogeneous set of training instances (instances classified in the same, or in a very similar, set of classes) is left to be covered, the fitness value can be reduced to 0. This happens when a rule covering all instances is induced. Thus, a rule covering all remaining instances belonging to a same/similar set of classes will have its fitness function set to 0. This occurs because $\frac{|S_{\neg r}|}{|S|} \times var(S_{\neg r})$ will be 0, and the values of $var(S)$ and $\frac{|S_r|}{|S|} \times var(S_r)$ will have the same value.



(a) Pheno (b) Spo

Fig. 8. Results for the Pheno and Spo datasets.



(a) Expr (b) Seq

Fig. 9. Results for the Expr and Seq datasets.

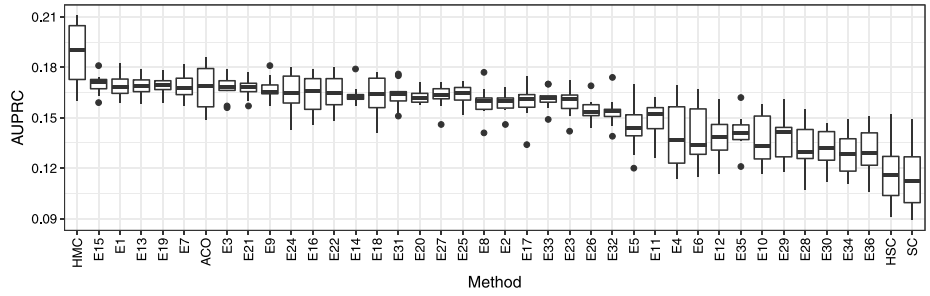


Fig. 10. $AU(\overline{PRC})$ results for all datasets and experiments.

Still considering relational tests, the representation of individuals is another characteristic that may also have harmed the HMC-GA performance when using relational tests. Recall that each 4-tuple in the antecedent of a rule is associated with an attribute. Thus, when an attribute A_i, associated to a given 4-tuple, is used in a relational test, it cannot be used in a propositional test anymore, since position i was already set as used.

We also analyzed our results considering the number of induced rules. Fig. 11 shows, for each HMC-GA variant, the average number of rules generated. We also show the average number of rules generated by the *hm*Ant-Miner (ACO) method, and by the method Clus-HMC (HMC). We did not show the methods Clus-HSC and Clus-SC in the graphs because they produced a huge number of rules, varying from nearly 2000 and 10 000 rules. Thus, as they clearly do not generate interpretable models, we left them out this analysis. We sort the methods according to the ranking obtained by the Friedman test.

According to the results from Fig. 11, the HMC-GA configurations that induced the smallest number of rules were those from the experiments E19, E15 and E13. These are among the configurations that obtained the best $AU(\overline{PRC})$ values, as shown in Fig. 10. Thus, the results show that these configurations are able to generate the best rules in terms of performance and interpretability. In comparison with configuration E19, Clus-HMC induced the 16th smallest set of rules, and *hm*Ant-Miner induced the 20th smallest set of rules.

The configurations that induced the largest number of rules were the experiments E34, E36, E28, E30, E29, and E35. These configurations also obtained some of the worst average $AU(\overline{PRC})$ values observed in Fig. 10. Especially, looking at experiments E34,

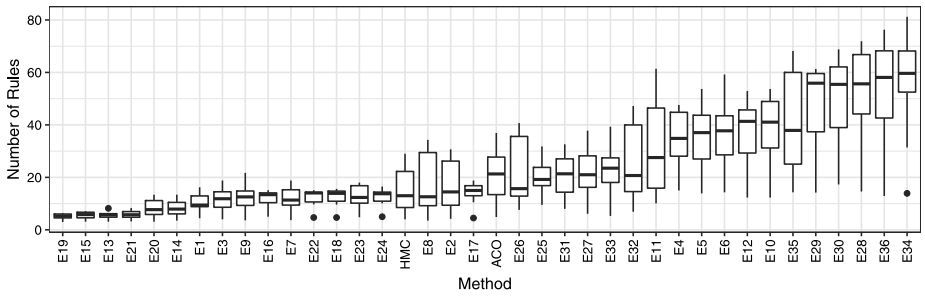


Fig. 11. Average number of rules for all datasets and experiments.

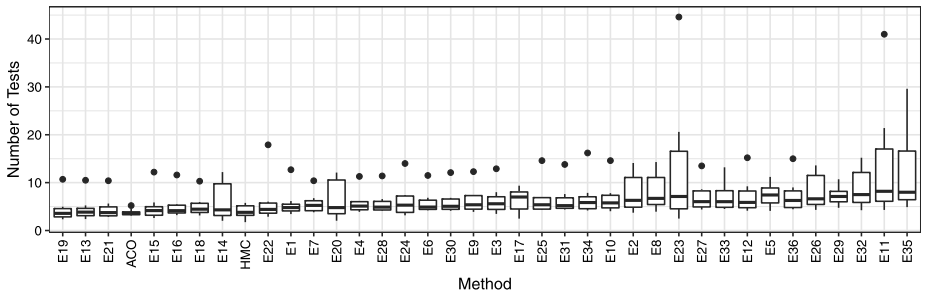


Fig. 12. Average number of tests per rule for all datasets and experiments.

E35 and E36, it is interesting to observe that all of them use a fitness function that does not consider the percentage of covered instances, which can directly impact the number of rules generated. The fitness function used considered the ponderation between variance gain and $AU(\overline{PRC})$ (F3). Besides, these experiments used the local search operator, which we have already observed, did not have a good influence on the method's performances. Thus, as these configurations generated a high number of rules, the type of rules generated (R1, R2, and R3) seemed not to have a great impact on the final result.

Continuing the analyses regarding the rules induced, we also analyzed the average size of these rules. Fig. 12 shows, for each HMC-GA experiment, the average number of tests per rule. We also show the average number of tests per rule generated by the *hm*Ant-Miner (ACO) and Clus-HMC (HMC) methods. Again, we sort the methods according to the ranking obtained by the Friedman test. Given that the number of rules generated by Clus-HSC and Clus-SC is extremely high, we did not compute their average number of tests. This is because even if they induce small rules, a model with a huge number of small rules is for sure less interpretable than a model with only a small number of small rules.

According to the results from Fig. 12, the smallest average number of tests per rule was obtained the HMC-GA configuration from Experiment 19 (E19). It generated an average of 4.27 tests per rule and was closely followed by the top ranking HMC-GA configurations in terms of $AU(\overline{PRC})$ values and number of rules. It generated fewer tests per rule than *hm*Ant-Miner and Clus-HMC.

Considering all characteristics investigated in this study to obtain a good set of rules, i.e., good $AU(\overline{PRC})$ values, and interpretability (small number of rules with a small number of tests each one), we can say that Experiment E19 seems to be the best configuration. It obtained one of the top $AU(\overline{PRC})$ values while generating the smallest sets of rules with the smallest average number of tests per rule.

In order to analyze what are the best individual configurations (fitness function, crossover operator, and type of induced rules), we observed the number of times these individual configurations appeared among the top 3, 5, and 7 best experimental results in terms of $AU(\overline{PRC})$ values, the average number of rules, and

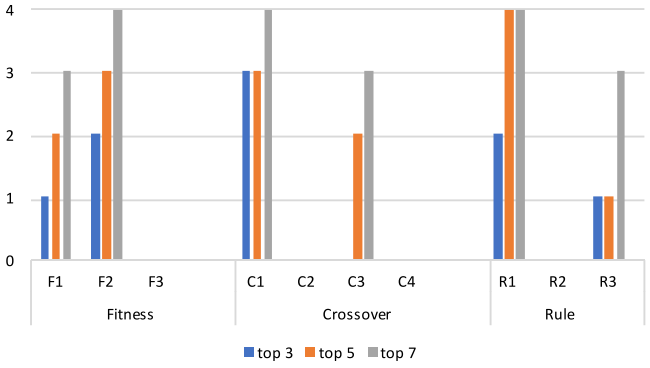


Fig. 13. Top individual configurations in terms of $AU(\overline{PRC})$ values.

the average number of tests per rule. These results are shown in Figs. 13–15, respectively.

Considering the $AU(\overline{PRC})$ values (Fig. 13), we can see that fitness function F2, crossover C1, and rule set R1 appeared two, three, and two times, respectively, considering the top 3 best experimental results. If we consider the top 5 best experimental results, configurations F2, C1 and R1 still appeared most of the times. Configuration C1, though, is closely followed by configuration C3 (three times against two times). The same pattern is observed if we analyze the top 7 experimental results (crossover C1 appears four times, against three times of crossover C3).

Looking at Experiments E19 (F2 × C3 × R1) and E13 (F2 × C1 × R1), which were considered, according to our analyses, the best experimental configurations in terms of performance and interpretability, it is possible to see that they contain the individual configurations that most appeared in the top best $AU(\overline{PRC})$ values. This enforces our previous observations, suggesting that (i) weighting the variance gain and the number of covered instances, (ii) performing our crossover without local search, and (iii) inducing rules with propositional tests, is the best configuration among those investigated.

Considering the average number of rules induced (Fig. 14), the fitness function F2 was the only one appearing in the top three and

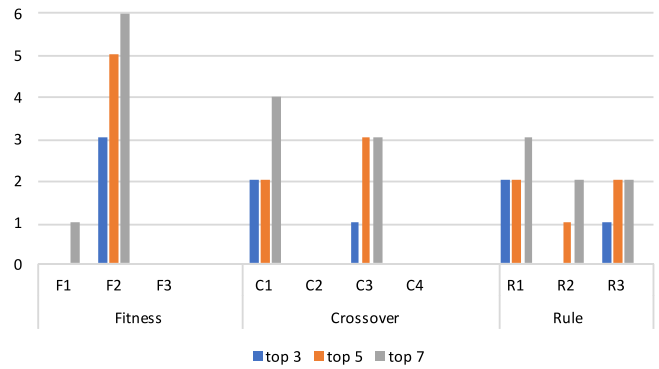


Fig. 14. Top individual configurations in terms of the average number of rules.

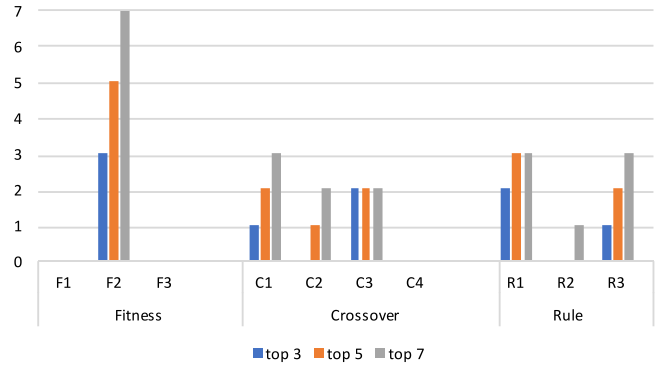


Fig. 15. Top individual configurations in terms of average number of tests per rule.

top five best experimental results. Considering the top seven best results, F2 was used in six of them. Thus, F2 is the best fitness function considering the number of rules generated. Considering the crossover operator, there was a balanced distribution of operations C1 and C3 among the top three, five and seven best results. Rules considering only relational tests (R2) did not appear among the top three best experimental results. Thus, the types of rule R1 and R3 are more recommended, since they have propositional tests. However, combined with configurations F2 and C1/C3, the types of rules generated apparently did not have a strong impact on the results.

Considering the average number of tests per rule (Fig. 15), a similar scenario to the one illustrated in Fig. 14 is observed. The fitness function F2 is the only configuration appearing in the top three, five and seven best results, and there is a balanced distribution considering crossover operators and types of rules. Again crossover C1 and C3 and rules R1 and R3 are the only ones appearing in the top three best experimental results.

Considering the overall best experiments, the results observed in all graphs from Figs. 13–15 support the fact that the best fitness function investigated considers variance gain and percentage of covered instances. The F2 fitness function leads to better results if combined with uniform crossover, considering distance or not, and without our local search operator. The best types of rules are those considering propositional tests in their antecedents.

5.1. Statistical analysis

This section presents the results of the statistical tests applied. We used the Friedman and Nemenyi tests with a confidence level of 95%. To evaluate the statistical significance of the results, we calculated the average Friedman rank for the 40 methods (ACO, Clus variations, and experiments E1 to E36) regarding $AU(\overline{PRC})$, number of rules, and number of tests.

Considering $AU(\overline{PRC})$, the average rank suggests that Clus-HMC is the best-ranked method. E15, the best evolutionary approach, figures as the second best method. The calculation of Iman's F statistic resulted in $F_f = 31.68$. The critical value of $F(k-1,(k-1)(n-1)) = F(39,351)$ for $\alpha = 0.05$ is 1.43. Since $F_f > F_{0.05}(39,351)$ $(31.68 > 1.43)$, the null hypothesis is rejected $(\text{p}{-}value = 2.2 * 10^{-16})$. We then proceed with a post-hoc Nemenyi test to find which method provides better results in a pairwise fashion. Fig. 16 shows the critical diagram for comparing the methods in terms of $AU(\overline{PRC})$. CD stands for the critical difference ($CD = 20.505$ at $\alpha = 0.05$), and methods connected by a line do not present statistically significant differences. Although none of the top-ranked methods outperform each other with statistical significance, it is possible to see that Clus-HMC, E15, E1, E13 and E19, for example, outperform more methods than the others (including ACO). The baselines HSC and SC are significantly outperformed by HMC, ACO and several of the evolutionary methods. Although Clus-HMC is best ranked then E15, it is important to note that this difference (5.1) is not statistically significant according to the Nemenyi's test, since $5.1 \ll 20.505$.

In terms of the number of rules, the average Friedman rank suggests that E19 is the best choice. Again, the calculation of Iman's F statistic resulted in $F_f > F_{0.05}(39,351)$ $(90.025 > 1.43)$, rejecting the null-hypothesis $(\text{p}{-}value = 2.20 * 10^{-16})$. Fig. 17 shows the critical diagram representing the results obtained by the Nemenyi's pairwise comparisons. Once again, there is one method (E19) that outperforms more methods (17) than others. ACO, for example, does not outperform any method with statistical significance, since it is connected to all methods in its right side of the diagram. Considering Clus-HMC, it statistically outperformed only six methods against 17 from the E19 variation.

Finally, the average Friedman rank suggests that E19 is also the best option regarding the number of tests per rule. Since $F_f > F_{0.05}(37,333)$ $(15.272 > 1.45)$, we reject the null-hypothesis $(\text{p}{-}value = 2.20 * 10^{-16})$ and proceed with a post-hoc Nemenyi test. According to Fig. 18, we can argue again that there is a method (E19) that outperforms more methods than the others investigated. Although there is no statistical difference between E19, ACO and Clus-HMC, E19 can be considered the best one, since (i) it outperforms 15 other methods against 12 from ACO and 2 from Clus-HMC; and (ii) E19 also outperforms ACO regarding $AU(\overline{PRC})$ and number of rules, and Clus-HMC regarding number of rules.

The results of the statistical tests enforce our previous observation, stating that the configuration of Experiment 19 (E19) is the best option among those investigated. According to the results, E19 is the best configuration in terms of interpretability, generating the smallest number of rules, and the smallest number of tests per rule. Also, E19 keeps good performance in terms of $AU(\overline{PRC})$, being among the top five best configurations.

5.2. Analysis for specific threshold values

In this section we present an analysis of our results using specific threshold values. Since the outputs of all methods have a probabilistic interpretation, we choose the values 0.2 and 0.5 as threshold values for illustration purposes. Thus, for 0.5, if the output of the methods for a given class is above or equal 0.5, we consider it as a positive prediction for the given class, and as a negative prediction otherwise. The same holds for the 0.2 value. We perform these evaluations to illustrate how all methods can be used as proper classifiers for a given task. With different threshold values, different results are obtained. Thus, suitable thresholds can be chosen depending on the desired outcome. For example, a specialist in a given domain could prefer models with high precision at

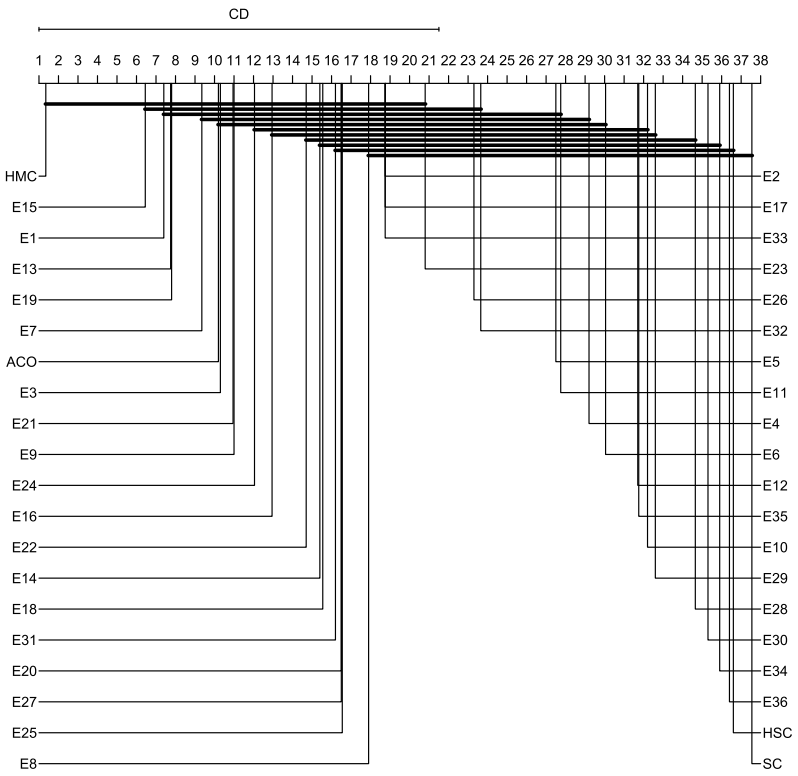


Fig. 16. Critical diagram considering the $AU(\overline{PRC})$ values.

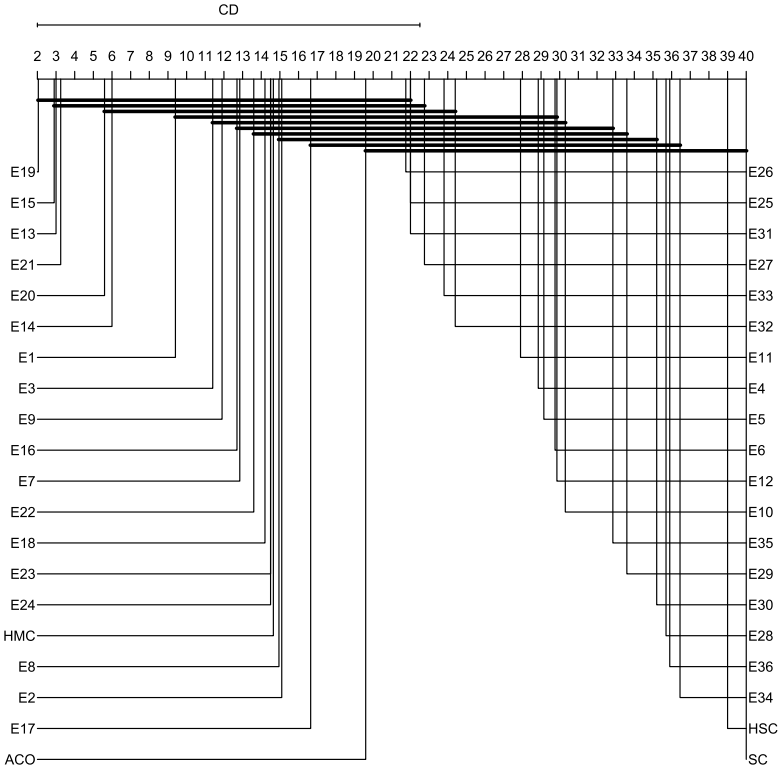


Fig. 17. Critical diagram considering the number of rules generated.

the cost of low recall or vice versa, or prefer models with maximal interpretability.

Figs. 19–21 present box-plots for Precision, Recall and F1-measure values obtained for all methods using a threshold value of 0.5. As can be seen, our previous recommended variation (Experiment E19) obtained one of the top three highest precision values (Fig. 19), while Clus-HMC remains the best classifier. However, this high precision comes with the cost of a low recall value (Fig. 20).

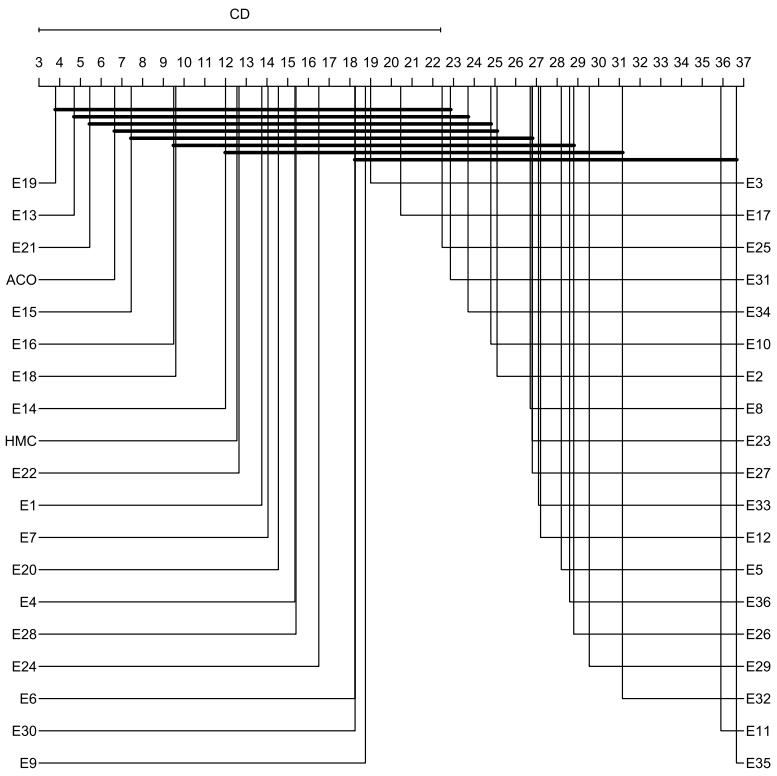


Fig. 18. Critical diagram considering the number of tests per rule.

The same behavior can be observed for the PCT-variations Clus-HMC, Clus-HSC and Clus-SC, i.e., higher precision values come at a cost of lower recall values, and vice versa.

The higher precision value obtained by variation E19 is expected since E19 is also among the top three variations regarding the smallest number of generated rules and the smallest number of tests per rule. Thus, it is expected that a more interpretable set of rules covers a smaller number of instances, being more precise, but at a cost of obtaining a smaller recall. If we look at the best recall values obtained by the GA variations (Fig. 20), we see that Experiments E34 and E28 are the top ones. These variations are also among the top three in the highest number of generated rules, thus being less interpretable. This is also expected since many rules cover a higher number of instances, but at the cost of a low precision (Fig. 19).

Thus, if a specialist is looking for an interpretable and precise model, we still can recommend the configuration of Experiment E19. Given this threshold value of 0.5, if one is looking for a configuration which best balances precision and recall, E34 is recommended (Fig. 21). However, this comes with the cost of generating a higher number of rules, thus reducing interpretability.

To illustrate how choosing a different threshold value can drastically change the results, Figs. 22–24 present box-plots for Precision, Recall and F1-measure values obtained for all methods using a threshold value of 0.2. As can be seen, the situation is now different, since, with this threshold, Clus-HMC has now the worst precision value (Fig. 22) and the highest recall value (Fig. 23). Regarding the GA variations, Experiment E19 still remains among the top three highest precision values, while Experiments E28 and E34 still obtained the best recall values. However, E19 now obtained a better balance between precision and recall than Experiments E28 and E34 (Fig. 24). Thus, if an expert is interested in the most interpretable model with a good balance between precision and recall, the configuration used in Experiment E19 can be still recommended. Of course, the user can be interested in the best possible balancing between precision and recall. In this case, the configuration of the Experiment E24 is recommended, but coming with the cost of losing interpretability.

As can be observed, there is a considerable variation in the results when changing threshold values from 0.5 to 0.2. Thus, thresholds can be chosen by a specialist depending on the context, focusing, for example, on high interpretability, high precision, high recall, or the best combination of precision and recall. Thus, if using the methods as proper classifiers for a given task, specific thresholds can be evaluated and chosen. When comparing models overall in many datasets, a threshold-independent evaluation is also adequate.

6. Conclusions and future works

This paper presented HMC-GA, a genetic algorithm for classification rule induction in hierarchical multi-label scenarios. Several experiments with different variations of HMC-GA were performed, considering combinations of different fitness functions, crossover operators, and types of rules generated. In total, 36 different experiments were performed with the genetic algorithm. HMC-GA was also compared with *hm*Ant-Miner, a natural computing-based method which generates HMC rules using an ant-colony optimization strategy. HMC-GA was also compared with three decision tree induction algorithms based on predictive clustering trees. They are state-of-the-art methods for HMC rule induction.

The experiments were carried out with ten freely available datasets organized according to the Funcat tree taxonomy. The datasets are related to protein function prediction, having features related to issues like phenotype and gene expression levels.

After analyzing the results, we concluded that the best configuration, among those investigated, is the configuration used in Experiment E19 (E19). E19 provides the most interpretable model, with the smallest number of rules and the smallest number of tests per rule, and still keeps its performance among the top best configurations. The configuration used in E19 combines (i) fitness function as ponderation between variance gain and percentage

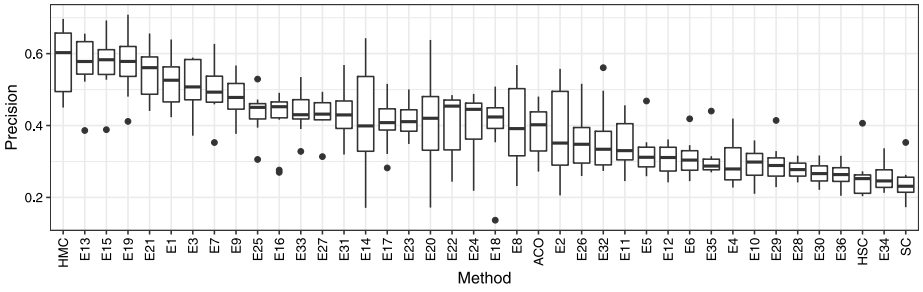


Fig. 19. Average precision values for all datasets and experiments for a threshold value of 0.5.

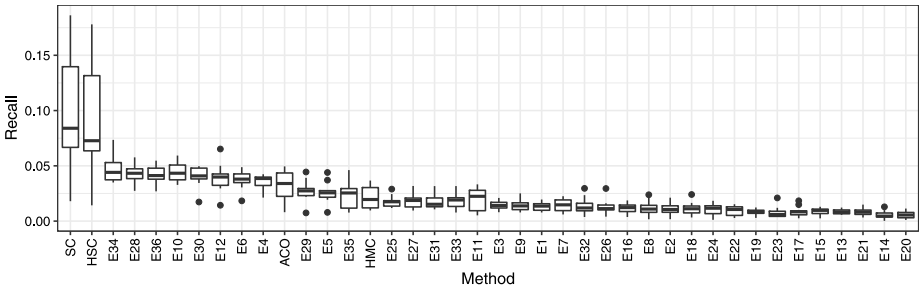


Fig. 20. Average recall values for all datasets and experiments for a threshold value of 0.5.

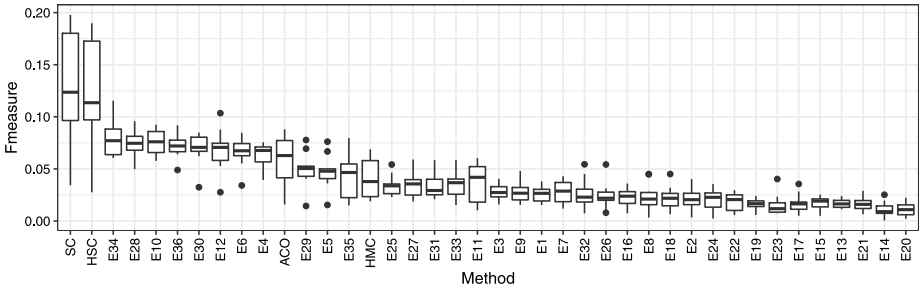


Fig. 21. Average F1 - measure values for all datasets and experiments for a threshold value of 0.5.

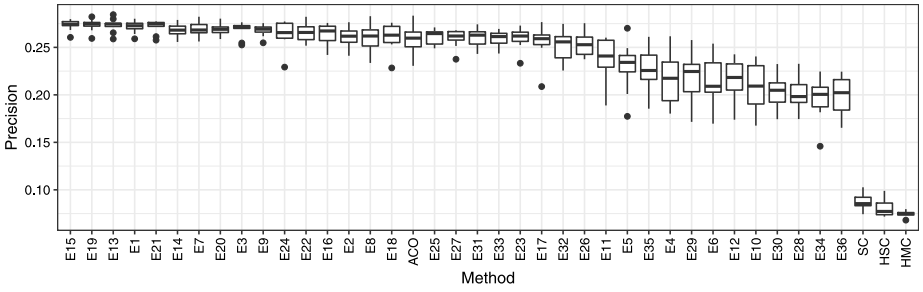


Fig. 22. Average precision values for all datasets and experiments for a threshold value of 0.2.

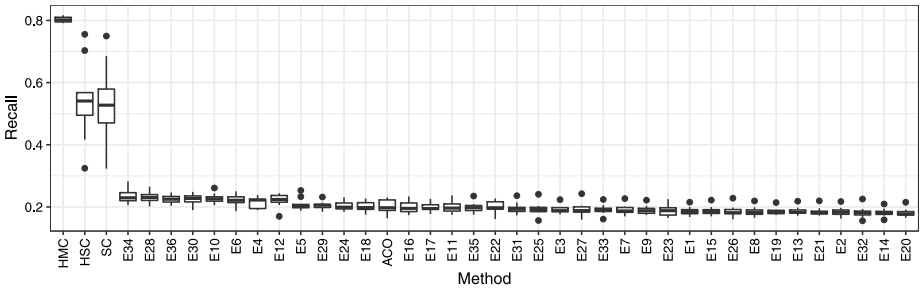


Fig. 23. Average recall values for all datasets and experiments for a threshold value of 0.2.

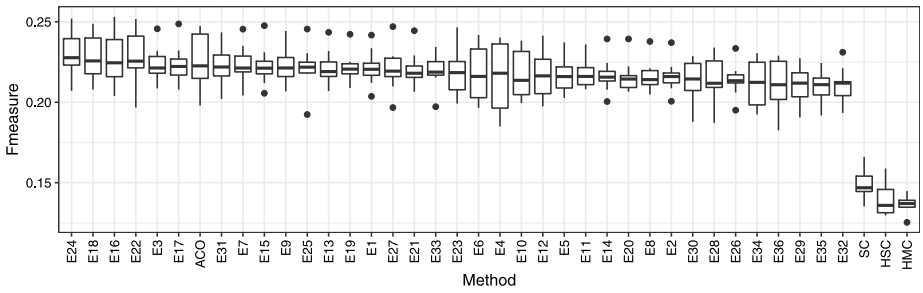


Fig. 24. Average F1 - measure values for all datasets and experiments for a threshold value of 0.2.

of covered instances (F2), (ii) distance-based uniform crossover without local search (C3), and (iii) only rules with propositional tests (R1).

As future work, we want to investigate the different HMC-GA configurations also on Gene Ontology structured datasets, which are organized in a much more challenging DAG structure. We also plan to explore multi-objective evolutionary optimization to improve our method, since many different objectives to be optimized are conflicting. Finally, we would like to investigate the performance of HMC-GA in other hierarchical and multi-label classification application domains, such as text and image classification.

Acknowledgments

The authors would like to thank Brazilian research agencies FAPESP, CNPq and CAPES for financial support, specially Coordenação de Aperfeiçoamento de Pessoal de Nível Superior-Brasil (CAPES) - Finance Code 001, and grants #2015/14300-1 and #2016/50457-5 - São Paulo Research Foundation (FAPESP), Brazil. The authors would also like to thank Intel Corporation.

References

[1] C. Silla, A. Freitas, A survey of hierarchical classification across different application domains, Data Min. Knowl. Discov. 22 (2010) 31–72.
[2] G. Valentini, True path rule hierarchical ensembles, in: International Workshop on Multiple Classifier Systems, 2009, pp. 232–241.
[3] S. Kiritchenko, S. Matwin, A.F. Famili, Hierarchical text categorization as a tool of associating genes with gene ontology codes, in: European Workshop on Data Mining and Text Mining in Bioinformatics, 2004, pp. 30–34.
[4] R. Cerri, R. Barros, A.C.P.L.F. Carvalho, Hierarchical multi-label classification using local neural networks, J. Comput. System Sci. 80 (1) (2013) 39–56.
[5] J. Wehrmann, R.C. Barros, S.N.d. Dôres, R. Cerri, Hierarchical multi-label classification with chained neural networks, in: Proceedings of the Symposium on Applied Computing (ACM SAC 2017), ACM, 2017, pp. 790–795.
[6] J. Wehrmann, R. Cerri, R.C. Barros, Hierarchical multi-label classification networks, in: International Conference on Machine Learning (ICML 2018), 2018, pp. 5225–5234.
[7] E.P. Costa, A.C. Lorena, A.C.P.L.F. Carvalho, A.A. Freitas, Top-down hierarchical ensembles of classifiers for predicting g-protein-coupled-receptor functions, in: Brazilian Symposium on Bioinformatics, in: LNBI, vol. 5167, Springer-Verlag, 2008, pp. 35–46.
[8] L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, S. Dzeroski, Predicting gene function using hierarchical multi-label decision tree ensembles, BMC Bioinformatics 11 (2010) 2.
[9] F. Otero, A. Freitas, C. Johnson, A hierarchical multi-label classification ant colony algorithm for protein function prediction, Memet. Comput. 2 (2010) 165–181.
[10] G. Valentini, True path rule hierarchical ensembles for genome-wide gene function prediction, IEEE/ACM Trans. Comput. Biol. Bioinform. 8 (3) (2011) 832–847.
[11] D. Stojanova, M. Ceci, D. Malerba, S. Dzeroski, Using PPI network autocorrelation in hierarchical multi-label classification trees for gene function prediction, BMC Bioinformatics 14 (1) (2013) 285.
[12] G. Yu, H. Zhu, C. Domeniconi, Predicting protein functions using incomplete hierarchical labels, BMC Bioinformatics 16 (1) (2015).
[13] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, H. Blockeel, Decision trees for hierarchical multi-label classification, Mach. Learn. 73 (2008) 185–214.
[14] R. Cerri, R.C. Barros, A.A. Freitas, A.C. de Carvalho, Evolving relational hierarchical classification rules for predicting gene ontology-based protein functions, in: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, in: GECCO Comp '14, ACM, New York, NY, USA, 2014, pp. 1279–1286.
[15] S. Dzeroski, N. Lavrac (Eds.), Relational Data Mining, Springer, 2001.
[16] A. Ruepp, A. Zollner, D. Maier, K. Albermann, J. Hani, M. Mokrejs, I. Tetko, U. Güldener, G. Mannhaupt, M. Münsterkötter, H.W. Mewes, The funcat, a functional annotation scheme for systematic classification of proteins from whole genomes, Nucleic Acids Res. 32 (18) (2004) 5539–5545.
[17] Z. Sun, Y. Zhao, D. Cao, H. Hao, Hierarchical multilabel classification with optimal path prediction, Neural Process. Lett. (2016) 1–15.
[18] R. Cerri, R.C. Barros, A.C.P.L.F. de Carvalho, Y. Jin, Reduction strategies for hierarchical multi-label classification in protein function prediction, BMC Bioinformatics 17 (1) (2016) 373.
[19] W. Bi, J. Kwok, Mandatory leaf node prediction in hierarchical multilabel classification, IEEE Trans. Neural Netw. Learn. Syst. 25 (12) (2014) 2275–2287.
[20] R. Baraniuk, V. Cevher, M. Duarte, C. Hegde, Model-based compressive sensing, IEEE Trans. Inform. Theory 56 (4) (2010) 1982–2001.
[21] H. Borges, J. Nievola, Multi-label hierarchical classification using a competitive neural network for protein function prediction, in: International Joint Conference on Neural Networks, 2012, pp. 1–8.
[22] N. Cesa-Bianchi, M. Re, G. Valentini, Synergy of multi-label hierarchical ensembles, data fusion, and cost-sensitive methods for gene functional inference, Mach. Learn. (2011) 1–33.
[23] N. Cesa-Bianchi, G. Valentini, Hierarchical cost-sensitive algorithms for genome-wide gene function prediction, J. Mach. Learn. Res. 8 (2010) 14–29.
[24] G. Valentini, M. Re, Weighted true path rule: A multilabel hierarchical algorithm for gene function prediction, in: Workshop on Learning from Multi-Label Data, held in ECML/PKDD, 2009, pp. 132–145.
[25] I. Triguero, C. Vens, Labelling strategies for hierarchical multi-label classification techniques, Pattern Recognit. 56 (C) (2016) 170–183.
[26] M. Ashburner, et al., Gene ontology: Tool for the unification of biology. The gene ontology consortium, Nature Genet. 25 (2000) 25–29.
[27] R. Carvalho, G. Brunoro, G. Pappa, HCGA: A genetic algorithm for hierarchical classification, in: IEEE Congress on Evolutionary Computation, 2011, pp. 933–940.
[28] M.-L. Zhang, Z.-H. Zhou, Multilabel neural networks with applications to functional genomics and text categorization, IEEE Trans. Knowl. Data Eng. 18 (2006) 1338–1351.
[29] I. Pillai, G. Fumera, F. Roli, Threshold optimisation for multi-label classifiers, Pattern Recognit. 46 (7) (2013) 2055–2065.
[30] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer-Verlag, Berlin, Heidelberg, 2002.
[31] J. He, X. Yao, Towards an analytic framework for analysing the computation time of evolutionary algorithms, Artificial Intelligence 145 (1) (2003) 59–97.
[32] M. Wilkins, E. Gasteiger, A. Bairoch, J. Sanchez, K. Williams, R. Appel, D. Hochstrasser, Protein identification and analysis tools in the expasy server, Methods Mol. Biol. (Clifton, N.J.) 112 (1999) 531–552, cited By (since 1996) 95.
[33] H.W. Mewes, et al., MIPS: A database for genomes and protein sequences, Nucleic Acids Res. 30 (2002) 31–34.
[34] A. Kumar, K.-H. Cheung, P. Ross-Macdonald, P.S.R. Coelho, P. Miller, M. Snyder, TRIPLES: A database of gene function in Saccharomyces cerevisiae, Nucl. Acids Res. 28 (1) (2000) 81–84.
[35] A. Clare, Machine Learning and Data Mining for Yeast Functional Genomics, Ph.D. thesis, University of Wales, 2003.
[36] P.T. Spellman, G. Sherlock, M.Q. Zhang, V.R. Iyer, K. Anders, M.B. Eisen, P.O. Brown, D. Botstein, B. Futcher, Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization, Mol. Biol. Cell 9 (12) (1998) 3273–3297.
[37] F.P. Roth, J.D. Hughes, P.W. Estep, G.M. Church, Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation, Nature Biotechnol. 16 (10) (1998) 939–945, http://dx.doi.org/10.1038/nbt1098-939.

[38] J.L. DeRisi, V.R. Iyer, P.O. Brown, Exploring the metabolic and genetic control of gene expression on a genomic scale, Science 278 (5338) (1997) 680–686, http://dx.doi.org/10.1126/science.278.5338.680.
[39] M.B. Eisen, P.T. Spellman, P.O. Brown, D. Botstein, Cluster analysis and display of genome-wide expression patterns, Proc. Natl. Acad. Sci. USA 95 (25) (1998) 14863–14868.
[40] A.P. Gasch, P.T. Spellman, C.M. Kao, O. Carmel-Harel, M.B. Eisen, G. Storz, D. Botstein, P.O. Brown, Genomic expression programs in the response of yeast cells to environmental changes, Mol. Biol. Cell 11 (12) (2000) 4241–4257.
[41] A.P. Gasch, M. Huang, S. Metzner, D. Botstein, S.J. Elledge, P.O. Brown, Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p, Mol. Biol. Cell 12 (2001) 2987–3003.
[42] S. Chu, J. Derisi, M. Eisen, J. Mulholl, D. Botstein, P.O. Brown, I. Herskowitz, The transcriptional program of sporulation in budding yeast, Science 282 (1998) 699–705.
[43] J. Davis, M. Goadrich, The relationship between Precision-Recall and ROC curves, in: International Conference on Machine Learning, 2006, pp. 233–240.
[44] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[45] H. Blockeel, L. De Raedt, J. Ramon, Top-down induction of clustering trees, in: International Conference on Machine Learning, 1998, pp. 55–63.
[46] D. Aleksovski, D. Kocev, S. Dzeroski, Evaluation of distance measures for hierarchical multilabel classification in functional genomics, in: Workshop on Learning from Multi-Label Data of ECML/PKDD, 2009, pp. 5–16.
[47] R. Iman, J. Davenport, Approximations of the critical region of the friedman statistic, Comm. Statist. (1980) 571–595.