Training Distributed GP Ensemble With a Selective Algorithm Based on Clustering and Pruning for Pattern Classification

Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano

***Abstract*—A boosting algorithm based on cellular genetic programming (GP) to build an ensemble of predictors is proposed. The method evolves a population of trees for a fixed number of rounds and, after each round, it chooses the predictors to include in the ensemble by applying a clustering algorithm to the population of classifiers. Clustering the population allows the selection of the most diverse and fittest trees that best contribute to improve classification accuracy. The method proposed runs on a distributed hybrid environment that combines the island and cellular models of parallel GP. The combination of the two models provides an efficient implementation of distributed GP, and, at the same time, the generation of low sized and accurate decision trees. The large amount of memory required to store the ensemble affects the performance of the method. This paper shows that, by applying suitable pruning strategies, it is possible to select a subset of the classifiers without increasing misclassification errors; indeed for some data sets, for up to 30% of pruning, ensemble accuracy increases. Experimental results show that the combination of clustering and pruning enhances classification accuracy of the ensemble approach.**

***Index Terms*—Boosting, classification, clustering, data mining, ensemble, genetic programming (GP).**

I. Introduction

Ensemble learning algorithms are an important topic of interest in the research community because of their capability of improving the classification accuracy of any single classifier. An ensemble of classifiers is constituted by a set of predictors that, instead of yielding their individual decisions to classify new examples, combine them together by adopting a strategy [4]–[6], [12], [19]. It has been pointed out that the boost in accuracy is tightly related to the diversity of the classifiers [12], [22]. Two classifiers are defined diverse if they make different incorrect predictions on new data points. Several approaches for building ensembles satisfying the diversity demand have been proposed. The *AdaBoost* algorithm introduced by Freund and Schapire [19] proved to be efficacious at generating different classifiers. It enables the underlying learning algorithm to focus on harder examples by adaptively changing the distributions of the training set on the base of the performance of previous classifiers.

The combination of genetic programming (GP) [21] and ensemble techniques has been receiving a lot of attention because of the improvements that GP obtains when enriched with these methods [10], [17], [20], [23], [25], [26], [30].

It is worth pointing out that an advantage of using GP, not yet exploited, is that a population of predictors could be considered as an ensemble of predictors. This assumption would provide at the same time many diverse classifiers. However, the size of a population generally is not small. Another problem could be the accuracy of some of the trees contained in the population. Taking all the individuals of a population at each generation is not a practical approach because of the resulting high number of predictors, possibly of low quality. A plausible proposal would be to use a clustering algorithm [13] to group individuals in the population that are similar with respect to a similarity measure, and then take the representatives of these clusters.

In this paper, a distributed boosting cellular GP classifier to build the ensemble of predictors is proposed. The algorithm, named (Clustering Boost Cellular Genetic Programming Classifier) ClustBoostCGPC, runs on a distributed environment based on a hybrid model [2] that combines the island model with the cellular model. The island model enables an efficient implementation of distributed GP. On the other hand, the cellular model allows the generation of classifiers with better accuracy and reduced tree size. Each node of the network is considered as an island that contains a learning algorithm, based on cellular GP, whose aim is to generate decision-tree predictors trained on the local data stored in the node. Every genetic program, however, though isolated, cooperates with the neighboring nodes by collaborating with the other learning components located on the network, and takes advantage of the cellular model by asynchronously exchanging the outermost individuals of the population.

ClustBoostCGPC constructs an ensemble of accurate and diverse classifiers by employing a clustering strategy to each subpopulation located on the nodes of the network. The strategy, at each boosting round, finds groups of individuals similar, with respect to a similarity measure, and then takes the individual of each cluster having the best fitness. This allows the selection, from each subpopulation, of the most dissimilar and fittest trees.

The main drawback of the approach proposed is that the size of the ensemble increases as the number of clusters and the nodes of the network increases. Thus, we could ask if it is possible to discard some of these predictors and still obtain comparable accuracy. This paper shows that, by applying suitable

Manuscript received June 1, 2006; revised December 1, 2006, and June 5, 2007. First published February 2, 2008; last published July 30, 2008 (projected). This work was supported in part by Regione Calabria (Programma Operativo Regionale—POR, Misura 3.16.B2 under Research Projects MIUR-legge 297/1999 and LOGICA Project.

The authors are with the Institute of High Performance Computing and Networking (ICAR), Italian National Research Council (CNR), University della Calabria, 87036 Rende, Italy (e-mail: folino@icar.cnr.it; pizzuti@icar.cnr.it; spezzano@icar.cnr.it).

Digital Object Identifier 10.1109/TEVC.2007.906658

pruning strategies, it is possible to select a subset of the classifiers without augmenting misclassification errors; indeed, for up to 30% of pruning, ensemble accuracy increases. The main contributions of the paper can be summarized as follows.

ClustBoostCGPC is a distributed ensemble method that mixes a supervised classification method with an unsupervised clustering method to build an ensemble of predictors.

Clustering the population of classifiers revealed a successful approach. In fact, the misclassification error rate of the ensemble sensibly diminishes when the ensemble is constituted by the best individuals in the clustered populations.

The method is enriched with pruning strategies that allow the reduction of the size of the ensemble, and, more notably, to improve classification accuracy. This result agrees with the logical principle of Occam's razor that one should not make more assumptions than the minimum needed and choose, from a set of equivalent models, the simplest one.

The algorithm runs on a distributed environment. The distributed architecture gives significant advantages in flexibility, extensibility, and efficiency since each node of the network works with its local data, and communicates the local model computed with the other nodes to obtain the results.

To assess the effectiveness of the method, experiments on several data sets are presented and compared with other approaches when different sizes of the ensemble are used.

Three pruning strategies are presented and compared to analyze their influence on the ensemble accuracy. The combination of these strategies is then investigated with the aim of decreasing ensemble size and improving classification accuracy. Experimental results pointed out that the proposed approach is particularly effective since it reduces the misclassification error rate of the algorithm.

This paper is organized as follows. The next section reviews ensemble techniques. In Section III, the ClustBoostCGPC algorithm and the software architecture used to run it are described. Section IV describes the pruning strategies adopted to reduce the size of the ensemble. In Section V, the results of the method on some standard problems are presented. Section VII, finally, concludes this paper by giving a discussion of the approach and some final considerations.

II. Ensemble Techniques

Let $S = \{(x_i, y_i) \mid i = 1, \ldots, N\}$ be a training set, where x_i called example or tuple or instance, is an attribute vector with m attributes and y_i is the class label associated with x_i. Let $\mathcal{X}$ denote the set of tuples and $\mathcal{Y}$ the set of class labels. Each attribute can be either discrete or continuous. A predictor (classifier), given a new example, has the task to predict the class label for it.

Ensemble techniques [4], [5], [12], [28] build a number of predictors, each on a different training set, then combine them together to classify the test set. Boosting was introduced by Schapire [28] and Freund [29] for boosting the performance of any "weak" learning algorithm, i.e., an algorithm that "generates classifiers which need only be a little bit better than random guessing" [29].

The boosting algorithm, called *AdaBoost*, adaptively changes the distribution of the training set depending on how difficult each example is to classify. Given the number T of trials (rounds) to execute, T weighted training sets $S_1, S_2, \ldots, S_T$ are sequentially generated and T classifiers $C^1, \ldots, C^T$ are built to compute a weak hypothesis $h_t : \mathcal{X} \rightarrow \mathcal{Y}$. Let w_i^t denote the weight of the example x_i at trial t. At the beginning, $w_i^1 = 1/n$ for each x_i. At each round $t = 1, \ldots, T$, a weak learner C^t, whose error ϵ^t is bounded to a value strictly less than 1/2, is built and the weights of the next trial are obtained by multiplying the weight of the correctly classified examples by $\beta^t = \epsilon^t/(1 - \epsilon^t)$ and renormalizing the weights so that $\Sigma_i w_i^{t+1} = 1$. Thus, "easy" examples get a lower weight, while "hard" examples, that tend to be misclassified, get higher weights. This induces *AdaBoost* to focus on examples that are hardest to classify. The boosted classifier gives the class label y that maximizes the sum of the weights of the weak hypotheses predicting that label, where the weight is defined as $ln(1/\beta^t)$.

Freund and Schapire [29] showed theoretically that *AdaBoost* can decrease the error of any weak learning algorithm and introduced two versions of the method *AdaBoost.M1* and *AdaBoost.M2*. *AdaBoost.M1*, when the number of classes is two, requires that the prediction be slightly better than random guessing. However, when the number of classes is more than 2, a more sophisticated error measure called *pseudoloss* is introduced. In this paper, we use the *AdaBoost.M1* version.

Proposals to combine GP and ensemble techniques can be found in [10], [17], [20], [23], [25], [26], and [30].

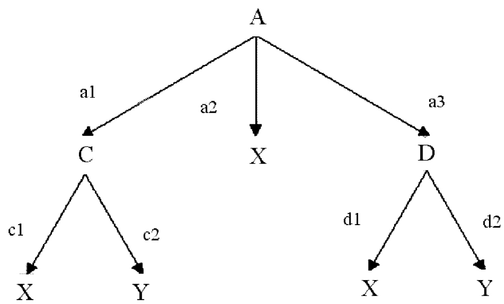
In particular, the Boost Cellular Genetic Programming Classifier (BoostCGPC) [17] implements the *AdaBoost.M1* boosting algorithm of Freund and Shapire [19] on a parallel computer by using the algorithm Cellular Genetic Programming for Data Classification (CGPC) [15] as base classifier. Given a training set S of size N and the number P of processors used to run the algorithm, BoostCGPC partitions the population of classifiers in P subpopulations, creates P subsets of tuples of size $n < N$ by uniformly sampling instances from S with replacement, and builds an ensemble of classification trees by choosing from each subpopulation the individual having the best fitness. In the next section, the algorithm ClustBoostCGPC is presented and the main differences with BoostCGPC are outlined.

III. ClustBoostCGPC

In this section, the description of the algorithm ClustBoostCGPC is given. The method builds an ensemble of classifiers by using, analogously to *BoostCGPC*, at each round of the boosting procedure, the algorithm CGPC [15] to create a population of predictors. However, instead of applying the Boost CGPC strategy of choosing from each subpopulation the individual having the best fitness, it finds k groups of individuals similar with respect to a similarity measure by employing the clustering algorithm k-means, and then takes the individual of each cluster having the best fitness. Before giving a detailed outline of the approach proposed, a brief review of the CGPC and k-means methods is provided.

A. The CGPC Algorithm

GP can be used to inductively generate a GP classifier as a decision tree for the task of data classification [21]. Decision trees, in fact, can be interpreted as composition of functions



(a)

Fig. 1. An example of the decision tree with terminal set $\mathcal{T} = \{X, Y\}$ and function set $\mathcal{F} = \{f_A(a_1, a_2, a_3), f_B(b_1, b_2), f_C(c_1, c_2), f_D(d_1, d_2)\}$.

where the function set is the set of attribute tests and the terminal set are the classes. The function set can be obtained by converting each attribute into an attribute-test function. For each attribute A, if $A_1, \ldots, A_n$ are the possible values A can assume, the corresponding attribute-test function f_A has arity n and if the value of A is A_i, then $f_A(A_1, \ldots, A_n) = A_i$. When a tuple has to be evaluated, the function at the root of the tree tests the corresponding attribute, and then executes the argument that results from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise, the new function is executed. Suppose a data set with two class labels X and Y, and attribute set $\{A(a_1, a_2, a_3), B(b_1, b_2), C(c_1, c_2), D(d_1, d_2)\}$, where $a_i, 1 \leq i \leq 3, b_j, c_j, d_j, j = 1, 2$ is the set of possible values A, B, C, D, respectively, can assume. Then, the terminal set is $\mathcal{T} = \{X, Y\}$ and the function set $\mathcal{F} = \{f_A(a_1, a_2, a_3), f_B(b_1, b_2), f_C(c_1, c_2), f_D(d_1, d_2)\}$. Fig. 1 shows a simple decision tree to decide if a tuple belongs to either the class X or Y. For example, if a tuple has the value of the attribute A equal to a_1 and that of C equal to c_1, then it is classified as X. Note that the nodes are labeled directly with the name of the attributes instead of the name of the associated function, for simplicity reasons. To evaluate the accuracy of the decision tree, the fraction of tuples classified into the correct class is computed. The fitness function [21] is defined as the number of training examples classified into the correct class. The CGPC algorithm used for data classification is described in Fig. 2.

CGPC adopts a cellular model of GP [27]. In the cellular model, each individual has a spatial location, a small neighborhood, and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a *panmictic* algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted. Cellular models of GP have been used to solve complex problems more accurately and with a minor number of iterations. Although fundamental theory is still an open research line, it has been empirically reported as being useful in maintaining diversity, and promoting slow diffusion of solutions through the grid. Part of their behavior is due to a lower selection pressure compared with that of panmictic GP.

CGPC generates a classifier as follows. At the beginning, for each cell, a random individual is generated (step 3) and its fitness

```
1.  Let p_c, p_m be crossover and mutation probability
2.      for each point i in grid do in parallel
3.      generate a random individual h_i
4.      evaluate the fitness of h_i
5.  end parallel for
6.  while not MaxNumberOfGeneration do
7.      for each point i in grid do in parallel
8.          generate a random probability p
9.          if (p < p_c)
10.             select the cell j, in the neighborhood of i,
                such that h_j has the best fitness
11.             produce the offspring by crossing h_i and h_j
12.             evaluate the fitness of the offsprings
13.             replace h_i with the best of the two offsprings
14.         else
15.             if ( p < p_m + p_c) then
16.                 mutate the individual
17.                 evaluate the fitness of the new h_i
18.             else
19.                 copy the current individual in the population
20.             end if
21.         end if
22. end parallel for
23. end while
```

Fig. 2. The CGPC algorithm.

is evaluated (step 4). The fitness is the number of training examples classified in the correct class. Then, at each generation, every tree undergoes one of the genetic operators (reproduction (step 19), crossover (steps 9–13), mutation (steps 15–17) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbor having the best fitness, and the offspring is generated. The current tree is then replaced by the best of the two offsprings if the fitness of the latter is better than that of the former. After the execution of the number of generations defined by the user, the individual with the best fitness represents the classifier.

B. The k-Means Algorithm for Clustering Classification Trees

The algorithm k-means [13] is a well-known clustering method that partitions a set of objects into k groups so that the intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured with respect to the mean value of the objects in a cluster, which can be considered as the cluster's center. The algorithm first randomly selects k objects, and assigns the remaining objects to the most similar cluster, where similarity is computed as the distance between the object and the center of the cluster. After that, the new mean values of the clusters are computed and this process is repeated until the criterion function converges. Typically, the squared-error criterion is used, defined as $E = \sum_{i=1}^{k} \sum_{p \in C_i} \mathrm{dist}(p, m_i)^2$, where E is the sum of square error for all the objects, dist is a distance measure, generally the Euclidean distance, p is an object, and m_i is the mean of cluster C_i. Both p and m_i are multidimensional objects.

In order to apply the k-means algorithm to a population of trees, it is necessary to specify the concept of distance between two individuals. To this end, we represent each classification tree h by a couple (f, e), where f is its fitness value and e is its distance from the empty tree Φ, considered as the origin tree. This representation allows us to take into account both the the concepts of phenotypic (i.e., based on fitness) and genotypic

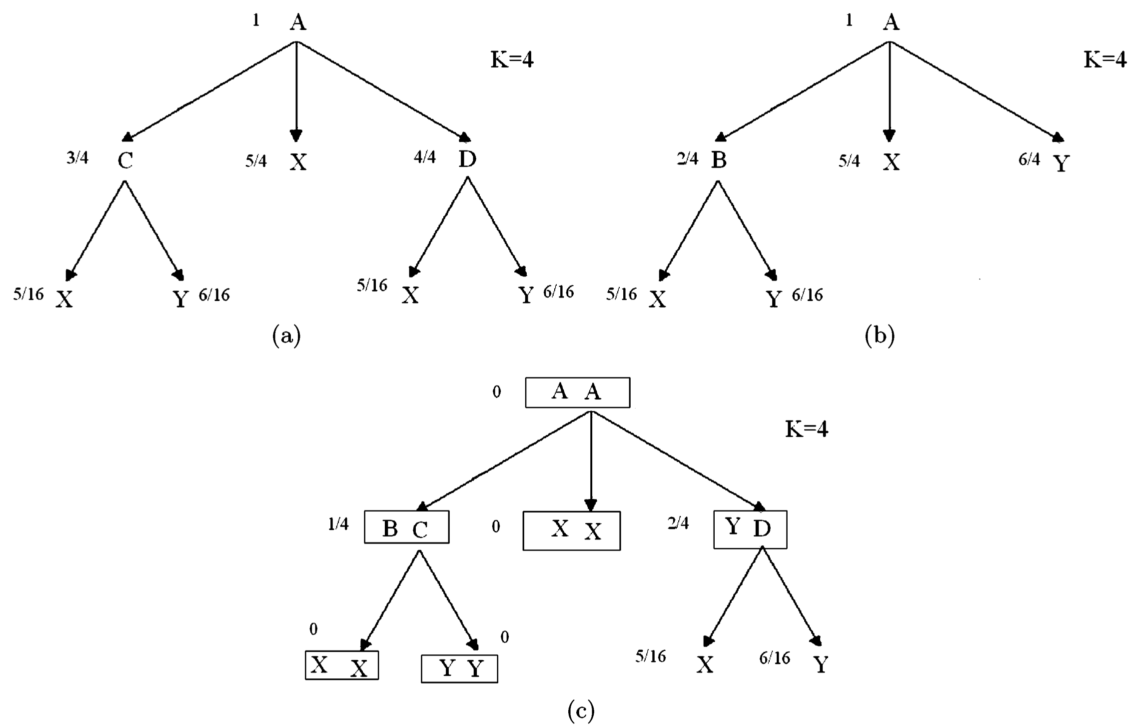


Fig. 3. (a) and (b) Two example trees and (c) the overlapped tree to compute their distance.

(i.e., based on the syntactical structure of individuals) diversity of the tree population [8]. The metric adopted to measure the structural distance between two genetic trees is that introduced by Ekárt and Németh [14].

The distance between two trees h_1 and h_2 is calculated in three steps: 1) h_1 and h_2 are overlapped at the root node and the process is applied recursively starting from the leftmost subtrees. 2) For each pair of nodes at matching positions, the difference of their codes (possibly raised to an exponent) is computed. 3) The differences computed in the previous step are combined in a weighted sum.

Formally, the distance of two trees h_1 and h_2 with roots R_1 and R_2 is defined as

$$\begin{aligned}\mathrm{dist}(h_1, h_2) \\ &= d(R_1, R_2) + \frac{1}{H}\sum_{i=1}^{m} \mathrm{dist}(\mathrm{child}_i(R_1), \mathrm{child}_i(R_2))\end{aligned}$$

where $d(R_1, R_2) = (|c(R_1) - c(R_2)|)^z$, c is a coding function $c : \{\mathcal{T} \cup \mathcal{F}\} \rightarrow \mathcal{N}$ that assigns a numeric code to each node of the tree, $\mathrm{child}_i(Y)$ is the ith of the m possible children of a generic node Y, if $i \leq m$, or the empty tree, otherwise. The constant H is used to give different weights to nodes belonging to different levels and z is a constant such that $z \in \mathcal{N}$.

For the example data set of the previous section, an encoding could be $\mathrm{c(A)} = 1, \mathrm{c(B)} = 2, \mathrm{c(C)} = 3, \mathrm{c(D)} = 4, \mathrm{c(X)} = 5, \mathrm{c(Y)} = 6$.

Fig. 3 shows two trees h_1 [Fig. 3(a)] and h_2 [Fig. 3(b)] with the weighted coding of each node, and their overlapping [Fig. 3(c)]. Corresponding nodes are enclosed in the rectangular boxes. The distance between h_1 and h_2, fixed $H = 4$ and $z = 1$, is computed as follows:

$$\begin{aligned}\mathrm{dist}(h_1, h_2) = (|1-1|) + \left(\left|\frac{3}{4} - \frac{2}{4}\right| + \left|\frac{5}{4}\right.\right. \\ \left.\left. - \frac{5}{4}\right| + \left|\frac{4}{4} - \frac{6}{4}\right|\right) + \left(\left|\frac{5}{16} - \frac{5}{16}\right| + \left|\frac{6}{16}\right.\right. \\ \left.\left. - \frac{6}{16}\right| + \left|\frac{5}{16} - 0\right| + \left|\frac{6}{16} - 0\right|\right) = 1.31.\end{aligned}$$

When computing the distance between a tree h and the empty tree $\Phi, \mathrm{dist}(h, \Phi)$ gives simply a weighted sum of the codes associated with the attributes appearing in the tree.

Once for each tree, the couple (f, e) has been computed, since both f and e are numbers, the k-means algorithm employs the Euclidean distance to the tree population by using this two dimensional representation.

C. The Distributed ClustBoostCGPC Algorithm to Build GP Ensemble

ClustBoostCGPC is a new ensemble learning algorithm for constructing GP ensembles. The idea is to incorporate different GP classifiers, each trained on different parts or aspects of the training set, so that the ensemble can learn from the whole training data. ClustBoostCGPC applies the boosting technique in a distributed hybrid model of parallel GP and uses a clustering-based selective algorithm to maintain the diversity of the ensemble by choosing in each population the most accurate predictors of each group.

Our approach aims to emphasize the cooperation among the individuals of the population (classifiers) using a hybrid model of parallel GP. It combines the island and cellular models of GP to enhance accuracy and to reduce performance fluctuation of the programs produced by GP. We used a hybrid model essentially for two reasons. First, the island model represents the best distributed implementation of GP that makes use of the domain decomposition technique. Second, the cellular model in each island allows the generation of classifiers with better accuracy and reduced tree size.

The island model is based on subpopulations created by dividing the original population into disjunctive subsets of individuals, usually of the same size. Each subpopulation is assigned to an island and a standard (panmictic) GP algorithm is executed on it. Occasionally, the migration process between subpopulations is carried out after a fixed number of generations. The hybrid model modifies the island model by substituting the standard GP algorithm with a cellular GP algorithm [16]. The introduction of the cellular approach improves the exploration capabilities of the algorithm because of a lower selection pressure that promotes a slow diffusion of solutions through the grid. In our model, we use the CGPC algorithm in each island. Each island operates in parallel on a subset of the tuples of the training set. The training and combination of the individual classifiers are carried together in the same learning process by a cooperative approach. Our model is based on the coevolution of different subpopulations of classifiers and a migration process that transfers asynchronously individuals among subpopulations.

In order to improve the prediction accuracy achieved by an ensemble, we need to ensure accuracy of classifiers and diversity among them. Although GP does not require any change in a training data to generate individuals of different behaviors, in [17], it is shown that GP enhanced with a boosting technique improves both the prediction accuracy and the running time with respect to the standard GP. ClustBoostCGPC combines the boosting method and the distributed hybrid model of GP to iteratively build an ensemble of classification trees through a fixed number of rounds.

The selection at each round of classifiers satisfying both high diversity and accuracy requirements is a difficult optimization task. To this end, in ClustBoostCGPC, we applied a method that gradually achieves diversity and accuracy. First, we employ the k-means clustering algorithm to divide all individuals of each subpopulation into groups (clusters) according to similarity of the classifiers. Then, the most accurate individual in each group, i.e., that having the best fitness value is selected.

A more formal description of the algorithm in pseudocode is shown in Fig 4. Let a network of P nodes be given, each having a training set S_j of size n_j. At the beginning, for every node $N_j, j = 1, \ldots, P$, a subpopulation Q_j is initialized with random individuals and the weights of the training instances are set to $1/n_j$ (steps 1–4). Each subpopulation Q_j is evolved for a fixed number of generations (step 7) and trained on its local training set S_j by running a copy of the CGPC algorithm (Fig. 2). After that, the evolved population of trees is clustered by using the k-means algorithm [13] and k groups of individuals are determined (step 8). For each group, the tree having the best fitness is chosen as representative of the cluster and output as the hypothesis computed. Then, the k individuals of each subpopulation are exchanged among the P nodes (step 9) and constitute the ensemble of predictors used to determine the weights of the

Given a network constituted by P nodes, each having a data set S_j of size n_j
1. **For** j = 1, 2, …, P (for each island in parallel)
2. **Initialize** the weights $w_i^1 = \frac{1}{n_j}$ for $i = 1, \ldots, n_j$,
   where n_j is the number of training examples on each node j.
3. **Initialize** the subpopulation Q_j, for $j = 1, \ldots, P$ with random individuals
4. **end parallel for**
5. **For** t = 1,2,3, …, T
6. **For** j = 1, 2, …, P (for each island in parallel)
7. **Train** *CGPC* on the partition S_j using a weighted fitness according to the distribution w^t
8. **Run** the k-means algorithm to compute k weak hypotheses $h_{j_1,t} \ldots h_{j_k,t} : \mathcal{X} \rightarrow \mathcal{Y}$
9. **Exchange** the hypotheses $h_{j_c,t}$ $c = 1, \ldots, k$ among the P nodes
10. **if** arg max $h_{j_c,t}(x_i) \neq y_i$
11. $D_{j_c} = 1$ else $D_{j_c} = 0$
12. **Compute** the error $\epsilon_{j_c}^t = \sum_{i=1}^n w_i^t D_{i_c}$
13. **Set** $\beta_{j_c}^t = \epsilon_{j_c}^t / (1 - \epsilon_{j_c}^t)$,
14. **Compute** $avg\_beta_j^t = \frac{\sum_{c=1}^k \beta_{j_c}^t}{k}$
15. **Update** the weights $w_i^{t+1} = avg\_beta_j^t \times w_i^t$ **if** $h_{j,t}(x_i) = y_i$
16. **end parallel for**
17. **end for t**
18. **output the hypothesis** :
20. $h_f = arg\ max\ (\sum_{t=1}^T \sum_{j=1}^P \sum_{c=1}^k log(\frac{1}{\beta_{j_c}^t}) D_{j_c}^t)$
21. where $D_{j_c}^t = 1$ if $h_{j_c,t}(x_i) = y_i$, 0 otherwise

Fig. 4. The ClustBoostCGPC algorithm.

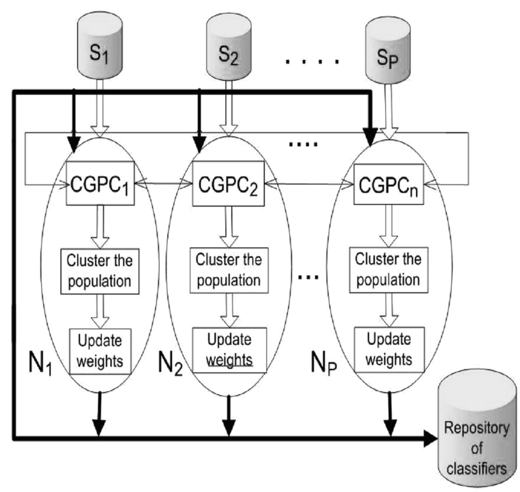


Fig. 5. Software architecture of ClustBoostCGPC.

examples for the next round. The error ϵ^t is computed by summing the weights of the misclassified tuples (steps 10–12). The weights for the next trial (step 15) are obtained by multiplying the weight of the correctly classified examples by avg_beta, that is the mean of the $\beta^t = \epsilon^t/(1 - \epsilon^t)$ values (steps 13–14) of the k weak hypotheses. Since avg_beta is less than 1, "easy" examples (i.e., already correctly classified) get a lower weight, while "hard" examples that tend to be misclassified get higher weights. The boosted classifier gives the class label y that maximizes the sum of the weights of the weak hypotheses predicting that label, where the weight is defined as $ln(1/\beta^t)$ (step 20). Note that the higher the weight of a weak hypothesis, the lower the misclassification error rate of the corresponding classifier.

We implemented ClustBoostCGPC using a distributed infrastructure and a distributed framework to run GP. The software architecture of ClustBoostCGPC is illustrated in Fig. 5.

We used *dCAGE* (distributed cellular GP system) a distributed environment to run genetic programs by an island model, which is an extension of [16]. *dCage* has been modified to support the hybrid variation of the classic island model.

In the new implementation, to take advantage of the cellular model of GP, the islands are evolved independently using the CGPC algorithm, and the outermost individuals are asynchronously exchanged. The training sets $S_i, i = 1, \ldots, P$ assigned to each of the P islands can be thought of as portions of the overall data set. The size of each subpopulation $Q_i, i = 1, \ldots, P$ present on a node, must be greater than a threshold determined from the granularity supported by the processor. Each node, using a training set S_i and a subpopulation Q_i, implements a classifier process CGPC_i as a learning algorithm and generates a population of classifiers. *dCAGE* distributes the evolutionary processes (islands) that implement the classification models over the network nodes using a configuration file that contains the configuration of the distributed system. *dCAGE* implements the hybrid model as a collection of cooperative autonomous islands running on the various hosts within an heterogeneous network that works as a peer-to-peer system. The *Message Passed Interface* (MPI) library is used to allow cooperation among the islands. Each island employed as a peer is identical to each other. At each round, a *collector* process collects the GP classifiers from the other nodes, handling the fusion of the results on behalf of the other peers, and redistributes the GP ensemble for future predictions to all the network nodes.

The configuration of the structure of the processors is based on a ring topology and a classifier process is assigned to each. During the boosting rounds, each classifier process maintains the local vector of the weights that directly reflect the prediction accuracy on that site. At every boosting round, the hypotheses generated by each of these classifiers (CGPC_i in Fig. 5) are clustered by employing the standard k-means algorithm. Then, the most accurate classifier in each group is selected to be included in the ensemble of predictors.

Next, the ensemble built so far is broadcasted to each classifier process to locally recalculate the new vector of the weights and a copy of the ensemble is stored in a repository. After the execution of the fixed number T of boosting rounds, the classifiers stored in the repository are used to evaluate the accuracy of the classification algorithm.

It is worth pointing out that though ClustBoostCGPC and BoostCGPC build an ensemble of classifiers for the task of data classification, there are some main differences between the two approaches. ClustBoostCGPC is a distributed algorithm that runs the boosting technique on a hybrid model of parallel GP by combining the island and cellular models. Thus, it assumes that each node has its own population and its own data set and that the classification algorithm CGPC be trained on the local data there contained.

On the other hand, BoostCGPC implements the boosting technique on a parallel computer by adopting the parallel cellular model of GP. In this case, if the number of processors at disposal to run the algorithm is P, the population is partitioned in P subpopulations, one for each processor, and P subsets of tuples are created by uniformly sampling instances from the overall training set with replacement.

Another main difference regards the individuals selected for participating to the ensemble. After a number of generations, BoostCGPC chooses the predictor with the best fitness. ClustBoostCGPC, instead, applies the clustering algorithm to the population of trees and picks the individual of each cluster having the best fitness. Though this policy proves to be beneficial for the accuracy of the method, as experimental results show, it introduces a memory overhead. The next section suggests the use of pruning strategies that partially overcome this problem.

IV. Reducing the Size of the Ensemble

A drawback of the method proposed, and of the ensemble methods, in general, is the large amount of memory required to maintain the classifiers. In our case, the size of the ensemble increases as the number of clusters and the number of nodes of the network increase. Thus, we could ask if it is possible to discard some of the predictors generated and still obtain comparable accuracy. This approach is well known in the literature and it is called *pruning* [24] or *thinning* [3] the ensemble. Pruning the ensemble requires a strategy to choose the classifiers to remove. There is a general agreement that the predictors forming the ensemble have to be both diverse and accurate. A pruning policy, thus, identifies the most similar classifiers and removes them. The concept of similarity in this context plays a central role. In the Machine Learning community, diversity means that the predictors have to make independent classification errors, i.e., they disagree with each other. A disagreement measure used in [24] is, for example, the κ statistics [1]. In the GP community, the concept of diversity is perceived in a different way [7], [9]. In particular, it reflects the structural diversity of the genetic programs in a generation [14]. In this paper, we adopt different diversity measures to choose the trees to prune. In the experimental results, we compare them and we show that the ensemble can be quite substantially pruned without increasing misclassification errors; indeed, up to 30% of pruning, ensemble accuracy increases.

The first two diversity measures used are the pairwise distance between two trees (denoted *pairwise*), and the distance of a tree from the empty tree (denoted *origin*), introduced in Section III-B.

The third measure is the κ statistics, defined as follows. Given two classifiers h_i and h_j, where $h_i, h_j : \mathcal{X} \rightarrow \mathcal{Y}$, consider the following $|\mathcal{Y}| \times |\mathcal{Y}|$ contingency table M. For elements $a, b \in \mathcal{Y}$, define $M_{a,b}$ to contain the number of examples x in the training set for which $h_i(x) = a$ and $h_j(x) = b$.

If h_i and h_j give identical classifications, all nonzero counts will appear along the diagonal. If h_i and h_j are very different, then there should be a large number of counts off the diagonal.

Let

$$\Theta_1 = \frac{\sum_{a=1}^{|\mathcal{Y}|} M_{a,a}}{N}$$

be the probability that two classifiers agree, where N is the size of the training set and $|\mathcal{Y}|$ is the number of different classes. Also, let

$$\Theta_2 = \sum_{a=1}^{|\mathcal{Y}|} \left(\frac{\sum_{b=1}^{|\mathcal{Y}|} M_{a,b}}{N} \frac{\sum_{b=1}^{|\mathcal{Y}|} M_{b,a}}{N} \right)$$

TABLE I
DATA SETS USED IN THE EXPERIMENTS

Dataset	Attr.	Tuples	Classes
Adult	14	48842	2
Census	41	299285	2
Covtype	54	581012	7
Mammography	10	11183	2
Phoneme	5	5404	2
Satimage	36	6435	6
Segment	19	2310	7

be the probability that two classifiers agree by chance, given the observed counts in the table. Then, the κ measure of disagreement between classifiers h_i and h_j is defined as

$$\kappa(h_i, h_j) = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}.$$

A value of $\kappa = 0$ implies that $\Theta_1 = \Theta_2$ and the two classifiers are considered to be different. A value of $\kappa = 1$ implies that $\Theta_1 = 1$, which means that the two classifiers agree on each example.

Thus, a pruning strategy first computes the κ, *origin*, and *pairwise* measures, and then chooses the predictors to eliminate in the following way.

If the *origin* measure is used, the predictors h_i are ordered in increasing order of $\mathrm{dist}(h_i, \Phi)$ and eliminated by starting with that having the highest value until the pruning percentage fixed has been reached.

If the *pairwise* (κ) measure is adopted, the distance $\mathrm{dist}(h_i, h_j)(\kappa(h_i, h_j))$ values are ordered in increasing order. The pruning strategy eliminates the pairs (h_i, h_j) of classifiers having the lowest value of $\mathrm{dist}(h_i, h_j)$ (the highest value of $\kappa(h_i, h_j)$), i.e., the more similar, considering them in increasing order of *dist* (decreasing order of κ) until the pruning percentage fixed has been reached.

V. EXPERIMENTAL RESULTS

In this section, ClustBoostCGPC and BoostCGPC are compared with seven data sets. Two data sets (*Census* and *Covtype*) are from the UCI KDD Archive,[1] three (*Segment*, *Satimage*, and *Adult*) are taken from the UCI Machine Learning Repository,[2] one (*Phoneme*) is from the ELENA project,[3] and one (*Mammography*) is a research data set used in [11]. The size and class distribution of these data sets are described in Table I.

The experiments were performed using a network composed by 10 1.133 GHz Pentium III nodes having 2 Gb of memory, interconnected over high-speed LAN connections.

All results were obtained by averaging over 50 runs by using 70% of the data sets for training and the remaining 30% for testing. In order to do a fair comparison between ClustBoostCGPC and BoostCGPC, we used a network of ten nodes for both algorithms. The number T of rounds was 10, population size 100 on each node, number of generations 100 (for a total number of generations $100 \times 10 = 1000$), and number of clusters fixed for ClustBoostGPC 5 and 10. Thus, BoostCGPC generated 100 classifiers, while ClustBoostCGPC, in one run 500 predictors, and in the other run 1000 predictors. However, the size of the ensembles for ClustBoostCGPC is greater than that of BoostCGPC. To analyze how the former algorithm performs when the number of predictors in the ensemble is equal to that in the latter, when executing BoostCGPC, we considered the first 5 and 10 fittest individuals from each subpopulation at each round. In this way, we obtained another two ensembles of size 500 and 1000.

TABLE II
MAIN PARAMETERS USED IN THE EXPERIMENTS

Name	Value
max_depth_for_new_trees	6
max_depth_after_crossover	17
max_mutant_depth	2
grow_method	RAMPED
selection_method	GROW
crossover_func_pt_fraction	0.7
crossover_any_pt_fraction	0.1
fitness_prop_repro_fraction	0.1
parsimony_factor	0

A main difference between ClustBoostCGPC and BoostCGPC regards the partitioning of the training sets on the nodes of the network. ClustBoostCGPC runs on a distributed environment where it is supposed that each node has its own data set. In order to simulate this kind of situation, each data set has been equally partitioned among the ten nodes. Thus, each node contains 1/10 of the training set. BoostCGPC runs on a parallel computer, thus according to the sequential AdaBoost approach, it creates ten subsets of tuples of size 1/10 the overall training set by uniformly sampling instances with replacement. The parameters used for the experiments are shown in Table II.

The main objectives of the experiments have been to investigate the influence of the clustering approach on the accuracy when different number of clusters are chosen, and to analyze and compare the pruning strategies described in the previous section.

Regarding the first objective, ClustBoostCGPC has been executed by fixing the number of clusters to 5 and 10, thus by using an ensemble of 500 an 1000 predictors, and compared with BoostCGPC that uses an ensemble of 100, 500, and 1000 predictors obtained as explained above. Table III shows the classification errors of the two algorithms. The table shows that for all the data sets, the clustering strategy sensibly improves the accuracy of the method. For example, on the *Adult* data set, ClustBoostCGPC (five clusters) obtains an error of 14.749 instead of 17.231, 16.29, and 15.85 of BoostCGPC with ensemble size 100, 500, and 1000, respectively. The table points out that the clustering approach is meaningful because the choice of the best individuals in the clustered populations produces a much better result with respect to choosing either the best, or the best five, or the best ten classification trees. However, as the table shows, augmenting the number of clusters is no more beneficial because the reduction of the misclassification error rate is minimal.

Table IV compares ClustBoostCGPC (ensemble size 500) with the other well-known classifications methods C4.5, SVM, and their boosted versions. We used the implementations

[1] http://kdd.ics.uci.edu/

[2] http://www.ics.uci.edu/~mlearn/MLRepository.html

[3] ftp.dice.ucl.ac.be in the directory pub/neural/ELENA/databases

TABLE III
COMPARISON OF THE MISCLASSIFICATION ERROR RATE OF BOOSTCGPC AND CLUSTBOOSTCGPC WITH DIFFERENT ENSEMBLE SIZES

Dataset	BoostCGPC 100 classifiers	BoostCGPC 500 classifiers	BoostCGPC 1000 classifiers	ClustBoostCGPC 500 classifiers	ClustBoostCGPC 1000 classifiers
Adult	17.231 ± 0.164	16.29 ± 0.157	15.85 ± 0.164	14.749 ± 0.163	14.641 ± 0.161
Census	6.12 ± 0.045	5.76 ± 0.045	5.51 ± 0.046	4.695 ± 0.042	4.681 ± 0.041
Covtype	33.374 ± 0.468	32.64 ± 0.463	32.81 ± 0.455	30.85 ± 0.453	30.044 ± 0.442
Mammography	2.159 ± 0.099	1.89 ± 0.097	1.92 ± 0.097	1.309 ± 0.093	1.304 ± 0.091
Phoneme	19.121 ± 0.475	18.03 ± 0.460	18.38 ± 0.474	16.968 ± 0.462	16.918 ± 0.448
Satimage	21.398 ± 0.573	20.68 ± 0.566	20.83 ± 0.574	20.22 ± 0.552	20.185 ± 0.543
Segment	13.452 ± 0.421	13.01 ± 0.406	13.22 ± 0.428	12.127 ± 0.418	12.114 ± 0.417

TABLE IV
COMPARISON OF THE MISCLASSIFICATION ERROR RATE OF CLUSTBOOSTCGPC AND C4.5, SVM, AND THEIR BOOSTED VERSIONS

Dataset	ClustBoostCGPC	C4.5	BoostC4.5	SVM	BoostSVM
Adult	14.749	15.3	15.1	16.4	16.8
Census	4.695	4.9	4.8	6.2	6.1
Covtype	30.850	17.2	17	-	-
Mammography	1.309	1.7	1.8	1.8	1.8
Phoneme	16.968	20.9	20.4	17.7	17.5
Satimage	20.220	15.6	15.2	15.6	14.3
Segment	12.127	11.5	10.7	8.57	7.2

TABLE V
ERROR AND GAIN OF PRUNED CLUSTBOOSTCGPC WITH RESPECT TO UNPRUNED CLUSTBOOSTCGPC AND UNPRUNED BOOSTCGPC

		origin			*pairwise*			*kappa*		
		Err	*GainC*	*GainB*	*Err*	*GainC*	*GainB*	*Err*	*GainC*	*GainB*
Adult	10%	14.711	0.26%	14.62%	14.462	1.95%	16.07%	**14.734**	0.10%	14.49%
	20%	**14.971**	-1.51%	13.12%	**14.742**	0.05%	14.44%	15.042	-1.99%	12.70%
	50%	15.425	-4.58%	10.48%	16.032	-8.70%	6.96%	15.823	-7.28%	8.17%
	80%	19.281	-30.73%	-11.90%	22.469	-52.34%	-30.40%	23.856	-61.74%	-38.45%
Census	10%	**4.582**	2.40%	25.13%	**4.289**	8.64%	29.92%	**4.453**	5.15%	27.24%
	20%	4.744	-1.04%	22.49%	4.706	-0.24%	23.10%	4.768	-1.56%	22.09%
	50%	5.073	-8.06%	17.11%	4.944	-5.31%	19.22%	5.085	-8.32%	16.91%
	80%	7.901	-68.30%	-29.10%	7.542	-60.65%	-23.24%	7.988	-70.15%	-30.52%
Covtype	10%	30.679	0.55%	8.08%	30.128	2.34%	9.73%	**30.203**	2.10%	9.50%
	20%	**30.904**	-0.18%	7.40%	**30.355**	1.60%	9.05%	30.800	0.16%	7.71%
	50%	32.322	-4.77%	3.15%	30.952	-0.33%	7.26%	31.012	-0.53%	7.08%
	80%	34.188	-10.82%	-2.44%	33.681	-9.18%	-0.92%	33.955	-10.06%	-1.74%
Mamm.	10%	**1.297**	0.95%	39.93%	**1.223**	6.60%	43.35%	**1.265**	3.39%	41.41%
	20%	1.385	-5.77%	35.85%	1.307	0.18%	39.46%	1.407	-7.45%	34.83%
	50%	1.467	-12.04%	32.05%	1.421	-8.52%	34.18%	1.604	-22.50%	25.71%
	80%	2.302	-75.81%	-6.62%	2.208	-68.63%	-2.27%	2.420	-84.82%	-12.09%
Phoneme	10%	16.893	0.44%	11.65%	16.528	2.59%	13.56%	**16.349**	3.65%	14.50%
	20%	**17.185**	-1.28%	10.13%	**16.892**	0.45%	11.66%	17.002	-0.20%	11.08%
	50%	18.042	-6.33%	5.64%	18.312	-7.92%	4.23%	18.207	-7.30%	4.78%
	80%	19.207	-13.20%	-0.45%	20.209	-19.10%	-5.69%	19.389	-14.27%	-1.40%
Satimage	10%	20.184	0.18%	5.67%	20.003	1.07%	6.52%	20.010	1.04%	6.49%
	20%	20.240	-0.10%	5.41%	20.081	0.69%	6.15%	20.120	0.49%	5.97%
	50%	**20.512**	-1.44%	4.14%	**20.958**	-3.65%	2.06%	**20.601**	-1.88%	3.72%
	80%	22.845	-12.98%	-6.76%	22.696	-12.25%	-6.07%	22.904	-13.27%	-7.04%
Segment	10%	12.027	0.82%	10.59%	11.906	1.82%	11.49%	12.004	1.01%	10.76%
	20%	12.209	-0.68%	9.24%	12.140	-0.11%	9.75%	**12.233**	-0.87%	9.06%
	50%	**12.638**	-4.21%	6.05%	**12.643**	-4.25%	6.01%	12.901	-6.38%	4.10%
	80%	16.467	-35.79%	-22.41%	15.842	-30.63%	-17.77%	15.068	-24.25%	-12.01%

contained in the WEKA [31] open source software available at http://www.cs.waikato.ac.nz/ml/weka/. The table shows that ClustBoostCGPC outperforms the other approaches on four out of the seven data sets. Regarding *Covtype*, the algorithms SVM and boosted SVM implemented in WEKA were not able to give an answer because of the size of the data set.

Using an ensemble of 500 predictors instead of 100 needs a larger amount of memory to store all the classifiers. Thus, the improved accuracy is obtained at the cost of higher storage requirements. In the second set of experiments, we show that the ensemble can be substantially pruned without decreasing performance. To this end, we considered the ensemble of 500 predictors and we applied the pruning strategies described in the previous section.

Table V reports the results of the different pruning strategies for all the data sets. The percentages of pruning experimented are 10%, 20%, 50%, and 80% of the ensemble. The table reports in the column named Err the misclassification error rate of the ensemble pruned of the percentage showed in the corresponding row. In column *GainC*, the relative gain in percentage of the pruned ensemble with respect to the complete ensemble generated by ClustBoostCGPC. In column *GainB*, the relative gain in percentage of the pruned ensemble with respect to the ensemble generated by BoostCGPC. A positive value means that the misclassification error rate is diminished, while a negative one that it has increased. To statistically validate the results, we performed a two-tailed paired t-test at 95% confidence interval. The values in bold of the columns Err highlight the percentage

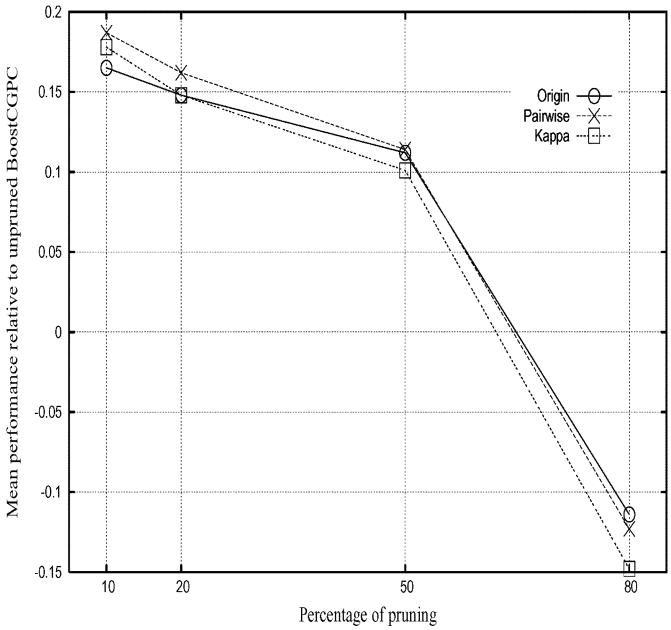


Fig. 6. Mean performance of pruned ClustBoostCGPC relative to unpruned ClustBoostCGPC with different pruning percentages.

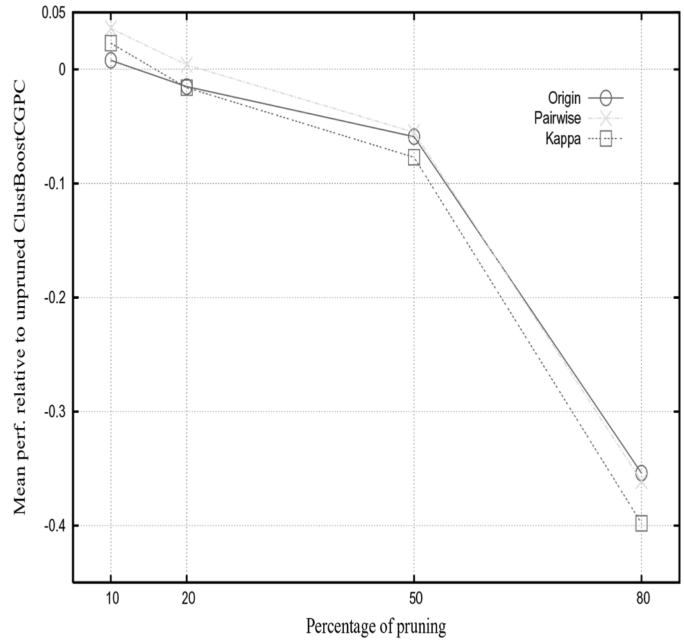


Fig. 7. Mean performance of pruned ClustBoostCGPC relative to unpruned BoostCGPC with different pruning percentages.

of training set needed by ClustBoostCGPC to obtain a lower error, meaningful with respect to the statistical test.

The table clearly shows that up to 50% of pruning, for all the data sets, independently the pruning strategy used, the ensemble can be reduced and still have an error lower than BoostCGPC (see column *GainB*). For example, the error obtained with the ensemble generated by ClustBoostCGPC on the *Census* data set pruned of 50% with the *pairwise* measure is 4.944, while that generated by BoostCGPC is 6.12, thus using the former approach gives a gain of 19.22%. It is worth noting that BoostCGPC with ensemble size 500 and 1000 obtains an error of 5.76 and 5.51, respectively, which is higher than the pruned ensemble of 250 predictors. Furthermore, pruning improves the performance of ClustBoostCGPC if 10% of the classifiers are eliminated. Indeed the *pairwise* strategy, for almost all the data sets, allows the pruning up to 20% of predictors and still decreases the misclassification error rate of ClustBoostCGPC. The structural diversity used in GP thus gives better results than the behavioral diversity employed in the Machine Learning community. This result could be explained by the observation that structural diversity means that the classification trees have nodes labelled with different attributes. As a consequence, the pruned ensemble is able to better generalize because of the presence of independent predictors.

Finally, Figs. 6 and 7 show the overall performances, averaged for all the data sets, in terms of the relative gain. In particular, Fig. 6 displays the relative performance of each pruning strategy computed as the difference between the corresponding misclassification error rate and that obtained by BoostCGPC divided by the gain, that is the difference in percentage points between these errors. A value of zero means that the pruned ensemble obtains the same performance as BoostCGPC, while a value greater than zero that the pruned ensemble performs better than BoostCGPC. Fig. 7 shows the same performance results compared with respect to ClustBoostCGPC. These two figures summarize the results of Table V and clearly point out that the pairwise strategy behaves better than the two others.

As already observed, Table V points out that independently, the pruning strategy, the deletion of 10% of trees enhance the accuracy of the method, while 20% of pruning is beneficial only for *pairwise* and, with regards to $kappa$, for some data sets. Thus, we wanted to verify whether the combination of the three strategies could produce better results than a single one. To this end, we deleted 20% of trees by choosing 10% of trees with one strategy and 10% with another, i.e., we combined *pairwise* and $kappa$, *pairwise* and *origin*, and $kappa$ and *origin*. Then, we deleted 30% of predictors by picking 10% of trees with respect to each pruning strategy. Table VI shows the result of this experiment. In the table p stands for *pairwise*, k for $kappa$, and o for *origin*. The results are very interesting. The deletion of 20% of trees from the ensemble by picking 10% with a strategy and another 10% with another strategy generates an error lower than both the ensemble pruned of 20% by applying one strategy and the unpruned ensemble. In particular, the error of the unpruned ensemble for the *Census, Mammography*, and *Phoneme* data sets diminishes from 4.695, 1.309, 16.968 to 4.421, 1.275, 16.265, respectively, when the *pairwise* and $kappa$ strategies are combined. For *Adult, Covtype, Satimage*, and *Segment* the reduction from 14.749, 30.850, 20.220, 12.127 to 14.726, 30.205, 20.049, 12.122, respectively, is obtained by mixing the *pairwise* and *origin* strategies. An error decrease of the pruned ensemble with respect to the unpruned one is attained also when 30% of classifiers are deleted by putting together the three strategies for the *Covtype, Phoneme*, and *Satimage* data sets. In any case, the elimination of 30% of predictors by mixing the $kappa$, *pairwise*, and *origin* strategies is always better than using one of them at a time. These experiments indicate that the ensemble techniques can achieve improved accuracy when good pruning policies are adopted.

TABLE VI
ERROR OF CLUSTBOOSTCGPC WITH RESPECT TO DIFFERENT PRUNING STRATEGIES

%. Prun.	*Strategy*	*Adult*	*Census*	*Covtype*	*Mamm.*	*Phoneme*	*Satimage*	*Segment*
	No Prun.	14.749	4.695	30.850	1.309	16.968	20.220	12.127
20%	*pairwise*	14.742	4.706	30.355	1.307	16.892	20.081	12.140
	kappa	15.042	4.768	30.800	1.407	17.002	20.120	12.333
	origin	14.971	4.744	30.904	1.385	17.185	20.240	12.209
	p + k	14.777	4.421	30.224	1.275	16.265	20.116	12.115
	p + o	14.726	4.588	30.205	1.287	16.564	20.049	12.122
	k + o	14.911	4.602	30.414	1.295	16.686	20.101	12.205
30%	*pairwise*	15.382	4.807	30.901	1.395	17.581	20.501	12.397
	kappa	15.264	4.946	30.871	1.459	17.481	20.322	12.508
	origin	15.152	4.885	31.12	1.407	17.433	20.389	12.359
	p + o + k	15.001	4.863	30.797	1.319	16.393	20.170	12.271

VI. DISCUSSION

A boosting algorithm based on cellular GP to build an ensemble of classifiers has been presented. The approach proposed presents two main novelties. The first is the application of a clustering algorithm to the subpopulations of the network nodes to build the ensemble. The second is the utilization of pruning strategies to discard some of the predictors but maintaining comparable accuracy. Both the ideas proved to be successful since the former allows the selection of the most diverse and fittest classification trees. The latter reduces the size of the ensemble, improving the classification accuracy. Experiments on several data sets showed that the choice of the fittest individual in the clustered populations produces a much better result with respect to choosing either the fittest or more than one fittest classification trees. ClustBoostCGPC has been compared with the state-of-the-art classification algorithms C4.5 and SVM. Results showed that ClustBoostCGPC outperforms the other approaches on four out of the seven data sets used. It is worth pointing out that the lower accuracy of ClustBoostCGPC with respect to the other two methods on the three multiclass data sets is due to the fact that ClustBoostCGPC implements the version *AdaBoost.M1* of the *AdaBoost* algorithm. As noted by Freund and Schapire [29], and experimented in [18], when the number of classes is more than two, and this is the case of *Covtype, Satimage*, and *Segment* data sets, *AdaBoost* needs a more sophisticated error measure that allows the weak learner focusing not only on the hard-to-classify examples, but also on the incorrect labels which are the hardest to discriminate. This error measure is implemented in the *AdaBoost.M2* version.

Taking more predictors from each subpopulation contained on the nodes of the network could give rise to criticisms because of the greater storage requirements necessary to maintain the ensemble. We showed that by employing suitable pruning strategies it is possible to select a subset of the classifiers without augmenting misclassification errors; indeed, up to 30% of pruning, ensemble accuracy increases.

VII. CONCLUSION

A distributed BoostCGPC has been presented. The method evolves a population of trees for a fixed number of rounds and, after each round, it chooses the predictors to include in the ensemble by applying a clustering algorithm to the population of classifiers. Pruning strategies to reduce ensemble size have also been studied. The method runs on a distributed environment based on a hybrid model that combines the island and cellular models of parallel GP. The combination of these two models provides an effective implementation of distributed GP, and the generation of classifiers with better accuracy and reduced tree size. A main advantage of the distributed architecture is that it enables for flexibility, extensibility, and efficiency since each node of the network works with its local data, and communicate with the other nodes, to obtain the results, only the local model computed, but not the data. Furthermore, this architecture is particularly apt to deal with the enormous amount of data that arrives in the form of continuous streams, generated in many application domains, such as credit card transactional flows, telephone records, sensor network data, network event logs. Future work aims at extending the ensemble approach to process these new kinds of data.

REFERENCES

[1] A. Agresti, *Categorical Data Analysis*. New York: Wiley, 1990.

[2] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans Evol. Comput.*, vol. 6, no. 5, pp. 443–462, Oct. 2002.

[3] R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "Ensembles diversity measures and their application to thinning," *Information Fusion*, vol. 6, pp. 49–62, 2005.

[4] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, pp. 105–139, 1999.

[5] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[6] L. Breiman, "Arcing classifiers," *Ann. Statistics*, vol. 26, pp. 801–824, 1998.

[7] E. Burke, S. Gustafson, and G. Kendall, "A survey and analysis of diversity measures in genetic programming," in *Proc. Genetic Evol. Comput. Conf. (GECCO 2002)*, 2002, pp. 716–723.

[8] E. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 47–62, 2004.

[9] E. Burke, S. Gustafson, G. Kendall, and N. Krasnogor, "Advanced population diversity measures in genetic programming," in *In Parallel Problem Solving from Nature—PPSN VII*. Granada, Spain: Springer-Verlag, 2002, vol. 2439, Lecture Notes in Computer Science, p. 341.

[10] E. CantúPaz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 1, pp. 54–68, Feb. 2003.

[11] N. Chawla, T. E. Moore, W. Bowyer K, L. O. Hall, C. Springer, and P. Kegelmeyer, "Investigation of bagging-like effects and decision trees versus neural nets in protein secondary structure prediction," in *Proc. BIOKDD01: Workshop on Data Mining Bioinformatics (SIGKDD 2001)*, 2001, pp. 50–59.

[12] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn*, vol. 40, pp. 139–157, 2000.

[13] R. C. Dubes and A. K. Jain, *Algorithms for Clustering Data*. Cambridge, MA: MIT Press, 1988.

[14] A. Ekárt and S. Z. Németh, "Maintaining the diversity of genetic programs," *Lecture Notes in Computer Science (EuroGP 2002)*, vol. 2278, pp. 162–171, 2002.

[15] G. Folino, C. Pizzuti, and G. Spezzano, "A cellular genetic programming approach to classification," in *Proc. Genetic Evol. Comput. Conf. (GECCO 1999)*, Orlando, FL, Jul. 1999, pp. 1015–1020.

[16] G. Folino, C. Pizzuti, and G. Spezzano, "A scalable cellular implementation of parallel genetic programming," *IEEE Trans. Evol. Comput.*, vol. 7, no. 1, pp. 37–53, Feb. 2003.

[17] G. Folino, C. Pizzuti, and G. Spezzano, "Boosting technique for combining cellular GP classifiers," in *Proc. 7th Eur. Conf. Genetic Program. (EuroGP 2004)*, M. Keijzer, U. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., Coimbra, Portugal, 2004, vol. 3003, LNCS, pp. 47–56.

[18] G. Folino, C. Pizzuti, and G. Spezzano, "GP ensembles for large-scale data classification," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 604–616, Oct. 2006.

[19] Y. Freund and R. Scapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.

[20] H. Iba, "Bagging, boosting, and bloating in genetic programming," in *Proc. Genetic Evol. Comput. Conf. (GECCO'99)*, Orlando, FL, Jul. 1999, pp. 1053–1060.

[21] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[22] L. I. Kuncheva and C. J. Whitaker, "Diversity measures in classifier ensembles," *Mach. Learn.*, vol. 51, pp. 181–207, 2003.

[23] W. B. Langdon and B. F. Buxton, "Genetic programming for combining classifiers," in *Proc. Genetic Evol. Comput. Conf. (GECCO 2001)*, San Francisco, CA, Jul. 2001, pp. 66–73.

[24] D. D. Margineantu and T. G. Dietterich, "Pruning adaptive boosting," in *Proc. Int. Conf. Mach. Learn.*, 1997, pp. 211–218.

[25] D. P. Pal and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 183–196, Feb. 2004.

[26] T. K. Paul, Y. Hasegawa, and H. Iba, "Classification of gene expression data by majority voting genetic programming classifier," in *Proc. IEEE World Congr. Comput. Intell.*, Vancouver, BC, Canada, 2006, pp. 8690–8697.

[27] C. C. Pettey, "Diffusion (cellular) models," in *Handbook of Evolutionary Computation*, D. B. F. Thomas Bäck and Z. Michalewicz, Eds. Bristol, U.K.: Oxford Univ. Press, 1997, pp. C6.4:1–6–, In.

[28] R. E. Schapire, "The strength of weak learnability," *Mach. Learn.*, vol. 5, no. 2, pp. 197–227, 1990.

[29] R. E. Schapire, "Boosting a weak learning by maiority," *Inf. Comput.*, vol. 121, no. 2, pp. 256–285, 1996.

[30] T. Soule, "Voting teams: A cooperative approach to non-typical problems using genetic programming," in *Proc. Genetic Evol. Comput. Conf. (GECCO'99)*, Orlando, FL, Jul. 1999, pp. 916–922.

[31] I. H. Witten and E. Frank, *Data Mining: Practical Mach. Learn Tools and Techniques*, 2nd ed. San Mateo, CA: Morgan Kaufmann.

Gianluigi Folino was born in Catanzaro, Italy, on May 27, 1972. He received the Laurea degree in engineering from the University of Calabria, Rende, Italy, in 1997.

In 1999, he joined Institute of High Performance Computing and Networking, Italian National Research Council (CNR-ICAR), supported by an INFN fellowship. Currently, he is a Researcher at the Institute of High Performance Computing and Networking (ICAR) of the Italian National Research Council (CNR) in the area of distributed and parallel computing and is also a Contract Professor with the Department of Computer Science, University of Calabria. He has published more than 50 papers in international conferences and journals. His research interests include cellular automata, genetic programming, swarm intelligence, peer-to-peer, and grid computing.

Clara Pizzuti received the Laurea degree in mathematics from the University of Calabria, Rende, Italy.

She is a Senior Researcher at the Institute of High Performance Computing and Networking (ICAR), Italian National Research Council (CNR). Since 1995, she has been a Contract Professor with the Department of Computer Science, University of Calabria. In the past, she worked in the research division of a software company on deductive databases, advanced logic-based systems, and abduction. She is serving as a program committee member of international conferences, and as a reviewer for several international journals. She has published more than 70 papers in conference proceedings and journals. Her research interests include knowledge discovery in databases, data mining, bioinformatics, evolutionary computation, genetic algorithms, and genetic programming.

Giandomenico Spezzano received the Laurea degree in industrial technologies engineering from the University of Calabria, Rende, Italy, in 1980.

He is a Research Director at the Institute of High Performance Computing and Networking (ICAR), Italian National Research Council (CNR), where he manages the Intelligent Grid and Peer-to-Peer Systems Group. He is also a Contract Professor with the Department of Electronics, Computer Science and Systems (DEIS), University of Calabria since 1994. Previously, he worked at the Consortium for Research and Applications of Information Technology (CRAI), Italy, where he has led various research projects in the distributed and parallel computing area. He has published two books and about 150 papers in conference proceedings and journals such as the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE COMPUTING AND SCIENCE ENGINEERING (CSE), *Future Generation Computer Systems* (FGCS), *Parallel and Distributed Computing Practices* (PDCP), and *Parallel Computing, Concurrency: Practice and Experience*. He is serving as a program committee member for many international conferences. His current research interests cover models and tools for massively parallel architecture, grid computing, peer-to-peer computing, parallel and distributed data mining, parallel genetic programming, cellular automata, and swarm intelligence.

Dr. Spezzano is a member of the Association for Computing Machinery (ACM) and the IEEE Computer Society.