Ensemble techniques for Parallel Genetic Programming based Classifiers

Gianluigi Folino, Clara Pizzuti and Giandomenico Spezzano

ICAR-CNR,
c/o DEIS, Univ. della Calabria
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@icar.cnr.it

Abstract. An extension of Cellular Genetic Programming for data classification to induce an ensemble of predictors is presented. Each classifier is trained on a different subset of the overall data, then they are combined to classify new tuples by applying a simple majority voting algorithm, like bagging. Preliminary results on a large data set show that the ensemble of classifiers trained on a sample of the data obtains higher accuracy than a single classifier that uses the entire data set at a much lower computational cost.

1 Introduction

Genetic programming (GP) [16] is a general purpose method that has been successfully applied to solve problems in different application domains. In the data mining field [8], GP has showed to be a particularly suitable technique to deal with the task of data classification [13, 19, 22, 17, 9, 10, 12] by evolving decision trees. Many data mining applications manage databases consisting of a very large number of objects, each having several attributes. This huge amount of data (gigabytes or even terabytes of data) is too large to fit into the memory of computers, thus it causes serious problems in the realization of predictors, such as decision trees [20]. One approach is to partition the training data into small subsets, obtain an ensemble of predictors on the basis of each subset, and then use a voting classification algorithm to predict the class label of new objects [5, 4, 6]. The main advantage of this approach is that accuracy comparable to that of a single predictor trained on all the training set can be obtained, but at a much lower computational cost.

Bagging [2] is one of the well known ensemble techniques that builds *bags* of data of the same size of the original data set by applying random sampling with replacement. It has been shown that bagging improves the accuracy of decision tree classifiers [2, 21]. Quinlan [21], for example, over 27 databases, experimented that bagging reduces the classification error by about 10% on average and that it is superior to C4.5 on 24 of the 27 data sets. However, when the data set is too large to fit into main memory, bags are also too large, thus, constructing and elaborating many bags of the same size of the entire data set is not feasible. In this

case data reduction through the partitioning of the data set into smaller subsets seems a good approach, though an important aspect to consider is which kind of partitioning has the minimal impact on the accuracy of results. Furthermore, to speed up the overall predictor generation process it seems straightforward to consider a parallel implementation of bagging.

In this paper we present an extension of Cellular Genetic Programming for data classification to induce an ensemble of predictors, each trained on a different subset of the overall data, and then combine them together to classify new tuples by applying a simple majority voting algorithm, like bagging. Preliminary results on a large data set show that the ensemble of classifiers trained on a subset of the data set obtains higher accuracy than a single classifier that uses the entire data set.

The paper is organized as follows. In section 2 a brief overview of the ensemble techniques is given. In section 3 the cellular parallel implementation of GP for data classification is presented. Section 4 proposes an extension of cellular genetic programming with ensemble techniques. In section 5, finally, the results of the method on some standard problems are presented.

2 Ensemble techniques

Let $S = \{(x_i, y_i)|i = 1, \ldots, n\}$ be a training set where x_i, called example, is an attribute vector with m attributes and y_i is the class label associated with x_i. A predictor, given a new example, has the task to predict the class label for it. Ensemble techniques build K predictors, each on a different subset of the training set, then combine them together to classify the test set.

Bagging (bootstrap aggregating) was introduced by Breiman in [2] and it is based on bootstrap samples (replicates) of the same size of the training set S. Each bootstrap sample is created by uniformly sampling instances from S with replacement, thus some examples may appear more than once while others may not appear in it. K bags $B_1, \ldots, B_K$ are generated and K classifiers $C_1, \ldots, C_K$ are built on each bag B_i. The number K of predictors is an input parameter. A final classifier classifies an example by giving as output the class predicted most often by $C_1, \ldots, C_K$, with ties solved arbitrarily.

More complex techniques such as boosting [14] and arching [3] adaptively change the distribution of the sample depending on how difficult each example is to classify. Bagging, boosting and variants have been studied and compared, and shown to be successful in improving the accuracy of predictors [7, 1]. These techniques, however, requires that the entire data sets be stored in main memory. When applied to large data sets this kind of approach could be impractical.

Breiman in [4] suggested that, when data sets are too large to fit into main memory, a possible approach is to partition the data in small pieces, build a predictor on each piece and then paste these predictors together. Breiman obtained classifiers of accuracy comparable if all the data set had been used. Similar results were found by Chan and Stolfo in [5]. In [6] Chawla et al. on a very large data set with a committee of eight classifiers trained on different partitions of

the data attained accuracy higher than one classifier trained on the entire data set.

Regarding the application of ensemble techniques to Genetic Programming, Iba in [15] proposed to extend Genetic Programming to deal with bagging and boosting. A population is divided in a set of subpopulations and each subpopulation is evolved on a training set sampled with replacement from the original data,. The size of the sampled training set is the same of the entire training set. Best individuals of each subpopulation participate in voting to give a prediction on the testing data. Experiments on some standard problems using ten subpopulations showed the effectiveness of the approach.

3 Data Classification using Cellular Genetic Programming

Approaches to data classification through genetic programming involve a lot of computation and their performances may drastically degrade when applied to large problems because of the intensive computation of fitness evaluation of each individual in the population. High performance computing is an essential component for increasing the performances and obtaining large-scale efficient classifiers. To this purpose, several approaches have been proposed. The different models used for distributing the computation and to ease parallelize genetic programming, cluster around two main approaches [23]: the well-known *island model* and the *cellular model*. In the island model several isolated subpopulations evolve in parallel, periodically exchanging by migration their best individuals with the neighboring subpopulations. In the cellular model each individual has a spatial location on a low-dimensional grid and the individuals interact locally within a small neighborhood. The model considers the population as a system of active individuals that interact only with their direct neighbors. Different neighborhoods can be defined for the cells and the fitness evaluation is done simultaneously for all the individuals. Selection, reproduction and mating take place locally within the neighborhood. In [11] a comparison of cellular genetic programming with both canonical genetic programming and the island model using benchmark problems of different complexity is presented and the the superiority of the cellular approach is shown.

Cellular genetic programming (CGP) for data classification was proposed in [9]. The method uses cellular automata as a framework to enable a fine-grained parallel implementation of GP through the diffusion model.The main advantages of parallel genetic programming for classification problems consist in handling large populations in a reasonable time, enabling fast convergence by reducing the number of iterations and execution time, favoring the cooperation in the search for good solutions, thus improving the accuracy of the method. The algorithm, in the following referred as $CGPC$ (*Cellular Genetic Programming Classifier*, is described in figure 1.

At the beginning, for each cell, an individual is randomly generated and its fitness is evaluated. Then, at each generation, every tree undergoes one of the

```
Let p_c, p_m be crossover and mutation probability
for each point i in grid do in parallel
    generate a random individual t_i
    evaluate the fitness of t_i
end parallel for
while not MaxNumberOfGeneration do
    for each point i in grid do in parallel
        generate a random probability p
        if (p < p_c)
            select the cell j, in the neighborhood of i,
            such that t_j has the best fitness
            produce the offspring by crossing t_i and t_j
            evaluate the fitness of the offspring
            replace t_i with the best of the two offspring
            evaluate the fitness of the new t_i
        else
        if ( p < p_m + p_c) then
            mutate the individual
            evaluate the fitness of the new t_i
        else
            copy the current individual in the population
        end if
        end if
end parallel for
end while
```

Fig. 1. The algorithm CGPC

genetic operators (reproduction, crossover, mutation) 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 string is then replaced by the best of the two offspring if the fitness of the latter is better than that of the former. The evaluation of the fitness of each classifier is calculated on the entire training data. After the execution of the number of generations defined by the user, the individual with the best fitness represents the classifier. The parallel implementation of the algorithm has been realized using a partitioning technique based upon a domain decomposition in conjunction with the Single-Program-Multiple-Data (SPMD) programming model. Furthermore, a parallel file system for partitioning the data set on different processors to obtain an efficient data access time was adopted. Figure 2 shows the software architecture of the implementation. On each processing element (PE) is allocated a process that contains a slice (SP_i) of the population and operates on all the data thought the parallel file system transferring the partitioned data set into the memory of the computer. In this way all the individuals of a subpopulation can operate on the training data without the need to request the transfer

of the data many times. The size of the subpopulation of each slice process is calculated by dividing the population for the number of processors of the parallel machine and ensuring that the size of each subpopulation be greater then a threshold determined from the granularity supported by the processor. For efficiency reasons, the individuals within a slice are combined into a single process that sequentially updates each individual. This reduces the amount of internal communication on each process, increasing the granularity of the application. Communication between processors is local, all that needs to be communicated between slices are the outermost individuals. The configuration of the structure of the processors is based on a ring topology and a slice process is assigned to each.

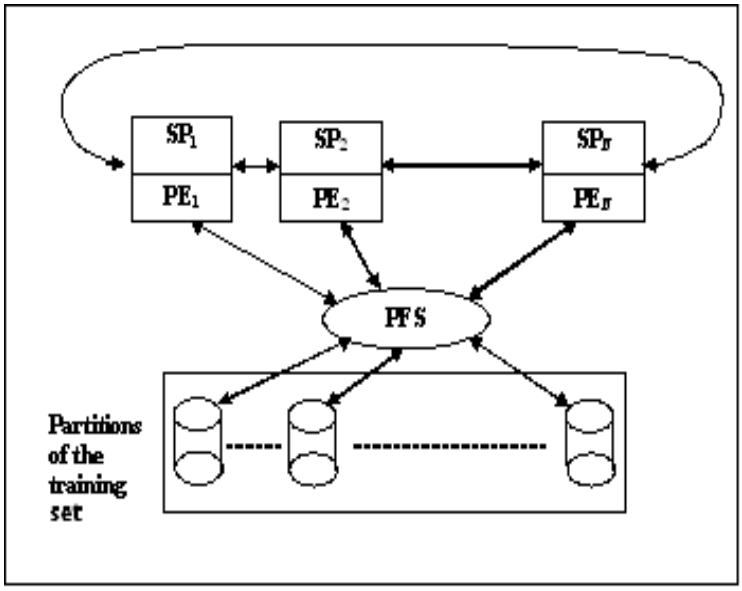

Fig. 2. Software architecture of CGPC

4 Ensemble of Classifiers in CGP

Although *CGPC* allows the construction of accurate decision trees, the performance of the algorithm is strongly depending on the size of the training set. In fact, in this model, one of the most expensive operation is the evaluation of the fitness of each decision tree: the entire data set is needed to compute the number of examples that are correctly classified, thus it must be replicated for each subpopulation. One approach to improve the performance of the model is to build an ensemble of classifiers, each working on a different subset of the original data set, then combine them together to classify the test set.

In this paper we propose an extension of $CGPC$ to generate an ensemble of classifiers, each trained on a different subset of the overall data and then use them together to classify new tuples by applying a simple majority voting algorithm, like bagging. The main feature of the new model, in the following referred as $BagCGPC$, is that each subpopulation generates a classifier working on a sample of the training data instead of using all the training set. The single classifier is always represented by the tree with the best fitness in the subpopulation. With K subpopulations we obtain K classifiers that constitute our ensemble. To take advantage of the cellular model of genetic programming subpopulations are not independently evolved, but they exchange the outmost individuals in an asynchronous way. Experimental results show that communication among the subpopulations produces an interesting positive result since the diffusion effect, that allows to transfer classifiers from a subpopulation to another, reduces the average size of trees and consequently improves the performances of the method since the evaluation time of the fitness is reduced.

This cooperative approach has the following advantages :

- samples of the data set are randomly generated;
- large data set that do not fit in main memory can be taken in consideration;
- the method is fault tolerant since the ensemble of classifiers has a collective fault masking ability operating with a variable number of classifiers.

Preliminary experiments on a large data set show that the ensemble of classifiers trained on a subset of the data set obtains higher accuracy than a single classifier the uses the entire data set.

Notice that our approach substantially differs from Iba's scheme [15] that extends genetic programming with bagging, since we use a parallel genetic programming model, we make cooperate the subpopulations to generate the classifiers and each subpopulation does not use the overall training set.

5 Experimental Results

In this section we present preliminary experiments and results of $BagCGPC$ on a large data set taken from the UCI Machine Learning Repository [18], the $Cens$ data set, and compare them with $CGPC$. The parameters used for the experiments are shown in table 1. Both algorithms run for 100 generations with a population size depending on the number of classifiers. We experimented $BagCGPC$ with 2, 3, 4, 5, 10, 15, and 20 classifiers. Every classifier, with its subpopulation, runs on a single processor of the parallel machine. The size of a subpopulation was fixed to 100, thus $CGPC$ used a population size of $100 \times number\ of\ classifiers$. For example, if an ensemble of 5 classifiers is considered, the population size for $CGPC$ is 500 (and the number of processors on which $CGPC$ is executed is 5) while, if the number of classifiers is 20, $CGPC$ used a population of 2000 elements (and executed on 20 processors). All results were obtained by averaging 10-fold cross-validation runs. The experiments were performed on a Linux cluster with 16 dual-processor 1,133 Ghz Pentium

III nodes having 2 Gbytes of memory connected by Myrinet and running Red Hat v7.2.

Table 1. Main parameter used in the experiments

Name	Value
max_depth_for_new_trees	6
max_depth_after_crossover	6
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

Table 2. Comparing accuracy for BagCGP and CGPC

Num. proc.	**BagCGP**				**CGPC**
	6000	**15000**	**30000**	**50000**	**All dataset**
1	5,992	5,831	5,770	5,687	5,582
2	5,823	5,779	5,638	5,404	5,407
3	5,662	5,516	5,428	5,375	5,349
4	5,536	5,372	5,254	5,205	5,278
5	5,439	5,338	5,108	5,072	5,244
10	5,359	5,215	5,068	5,040	5,211
15	5,340	5,207	5,060	5,028	5,185
20	5,322	5,183	5,020	5,004	5,127

In our experiments we wanted to investigate the influence of the sample sizes on the accuracy of the method. To this end we used the *Cens* data set, a large real data set containing weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. The data set consists of 299285 tuples, 42 attributes and two classes.

Figure 3, and the corresponding table 2, show the effect of different sample sizes on accuracy as the number of classifiers increases. For each ensemble, the error of *CGPC* and *BagCGPC* with sample size of 6000, 15000, 30000, and 50000 are shown. From the figure we can note that when the sample size is 6000, *BagCGPC* is not able to outperform the single classifier working on the entire data set. The same effects is obtained when the number of classifiers is less then

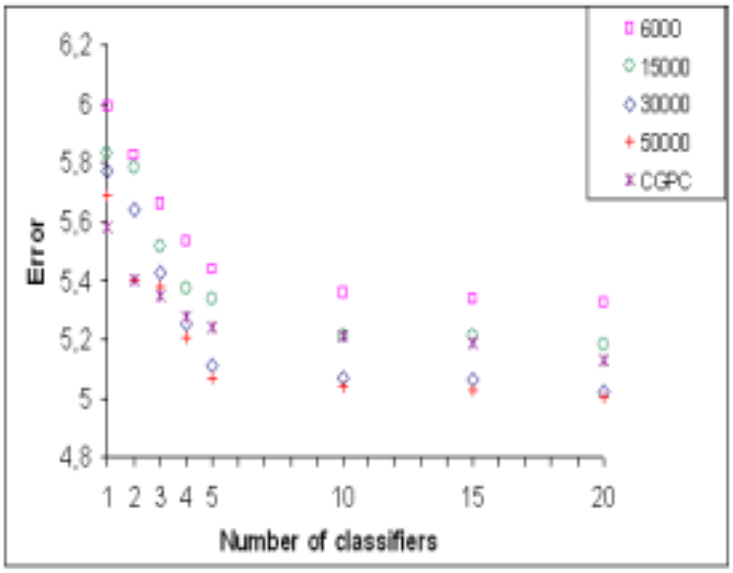

Fig. 3. Error for different sample sizes of training set vs number of classifiers used. (Cens dataset)

three. But, as the sample size or the number of classifiers increases, $BagCGPC$ is able to obtain an error lower than $CGPC$. An ensemble of four classifiers using a subset of the data of size 30000 obtains higher accuracy. Augmenting the sample size and the number of classifiers a further increase can be attained, though a saturation point stops this effect. Another positive result regards the computation time. In fact $BagCGPC$ is much more efficient than $CGPC$. Table 3 shows the the execution times of $CGPC$ and $BagCGPC$ for each sample. For example, $CGPC$ required 6053 seconds to run on the Cens data set for 100 generations with a population size of 500 elements. When five classifiers are employed, each using 50000 tuples and a population size of 100 elements, $BagCGPC$ needed 1117 seconds of computation time.

As already stated in the previous section, communication among the subpopulations has a positive impact on the average size of the trees and consequently improves the performances of the method since the evaluation time of the fitness is reduced. This effect can be seen in figure 4, where the average length of the trees is shown when the algorithm ran with 5 classifiers and 50000 tuple, in case of communication of the border trees among the subpopulations and no communication, respectively. After 100 generations, in the former case the average size is 900 and the computation time is 1117 seconds, as already said, while in the latter the average size is about 10000 and the time needed by the method was 4081 seconds. In the lack of communication also accuracy worsened, going from 5,07 to 5,56.

Table 3. Comparing execution times for BagCGP and CGPC

Num. proc.	BagCGP				CGPC
	6000	15000	30000	50000	All dataset
1	588	633	783	841	3760
2	596	654	823	976	4051
3	612	719	980	1022	4233
4	635	725	965	1056	5598
5	668	843	1064	1117	6053
10	799	902	1116	1278	6385
15	823	922	1226	1316	8026
20	964	1055	1456	1621	9161

Table 4. Comparing execution times and accuracy of *CGPC*, *BagCGPC* and *BagCGPC* without communication using 5 classifiers and 1/5 of the training tuples running on 5 processors

Dataset	CGPC		BagCGPC		BagCGPC without com.	
	Test Error	Time (sec)	Test Error	Time (sec)	Test Error	Time (sec)
Adult	17,26	717	16,58	209	18,26	364
Mushroom	0,35	143	0,43	52	1,23	80
Sat	23,04	228	23,63	30	24,23	47
Shuttle	5,18	729	5,37	151	8,54	174

A confirmation of this behavior was obtained for other 4 data sets of small-medium size, shown in table 4, where the execution time and the error on the test set for *CGPC* and *BagCGPC* (with and without communications) algorithms are reported, in case of 5 classifiers and a sample of size 1/5 the overall training set. In all the four data sets we obtained a lower execution time and a higher accuracy using the committee of classifiers generated by the *BagCGPC* algorithm with communication with respect to no communication, though in this case the misclassification error is not always lower than that of the single classifier generated by *CGPC*.

6 Conclusions and Future Work

An extension of Cellular Genetic Programming for data classification to induce an ensemble of predictors was presented. The approach is able to deal with large data set that do not fit in main memory since each classifier is trained on a subset of the overall training data. Preliminary experiments on a large real data set showed that higher accuracy can be obtained by using a sample of reasonable size at a much lower computational cost. The experiments showed that sample size influences the achievable accuracy and that, choosing a suitable sample size,

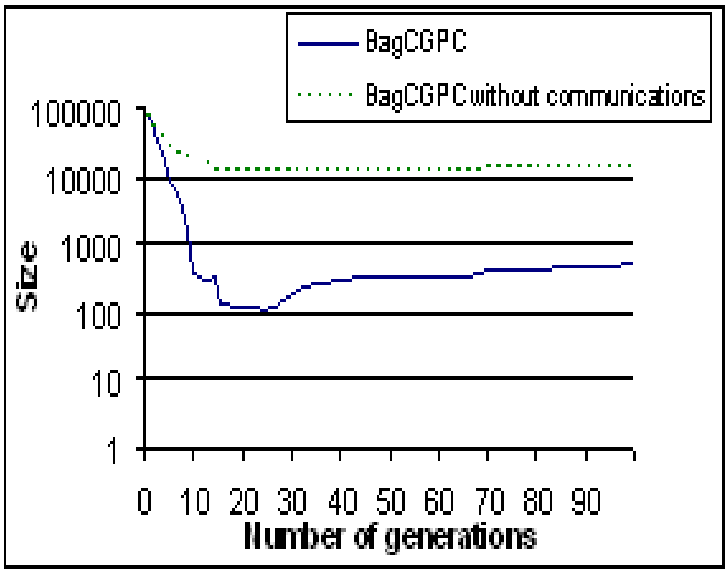

Fig. 4. Average length of the trees for the Cens dataset (5 classifiers, 50000 tuples) with and without communications.

a low number of classifiers is sufficient to obtain higher accuracy. Furthermore we showed that the sharing of information between the subpopulations improves the ability of algorithm to learn since trees with a smaller size are produced. The method proposed is fault tolerant since the ensemble of classifiers has a collective fault masking ability operating with a variable number of classifiers. We are planning an experimental study on a wide number of very large benchmark problems to substantiate the validity of the proposed approach.

References

1. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, (36):105–139, 1999.
2. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
3. Leo Breiman. Arcing classifiers. *Annals of Statistics*, 26:801–824, 1998.
4. Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1,2):85–103, 1999.
5. P. K. Chan and S.J. Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. In *International Conference on Machine Learning ICML95*, pages 90–98, 1995.
6. N. Chawla, T.E. Moore, W. Bowyer K, L.O. Hall, C. Springer, and P. Kegelmeyer. Bagging-like effects for decision trees and neural nets in protein secondary structure prediction. In *BIOKDD01: Workshop on Data mining in Bioinformatics (SIGKDD01)*, 2001.

7. Thomas G. Dietterich. An experimental comparison of three methods for costructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, (40):139–157, 2000.
8. U.M. Fayyad, G. Piatesky-Shapiro, and P. Smith. From data mining to knowledge discovery: an overview. In U.M. Fayyad & al. (Eds), editor, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI/MIT Press, 1996.
9. G. Folino, C. Pizzuti, and G. Spezzano. A cellular genetic programming approach to classification. In *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, pages 1015–1020, Orlando, Florida, July 1999. Morgan Kaufmann.
10. G. Folino, C. Pizzuti, and G. Spezzano. Genetic programming and simulated annealing: A hybrid method to evolve decision trees. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty, editors, *Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 294–303, Edinburgh, UK, 15-16 April 2000. Springer-Verlag.
11. G. Folino, C. Pizzuti, and G. Spezzano. Cage: A tool for parallel genetic programming applications. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 64–73, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.
12. G. Folino, C. Pizzuti, and G. Spezzano. Parallel genetic programming for decision tree induction. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence ICTAI01*, pages 129–135. IEEE Computer Society, 2001.
13. A.A. Freitas. A genetic programming framework for two data mining tasks: Classification and generalised rule induction. In *Proceedings of the 2nd Int. Conference on Genetic Programming*, pages 96–101. Stanford University, CA, USA, 1997.
14. Y. Freund and R. Scapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th Int. Conference on Machine Learning*, pages 148–156, 1996.
15. Hitoshi Iba. Bagging, boosting, and bloating in genetic programming. In *Proc. Of the Genetic and Evolutionary Computation Conference GECCO99*, pages 1053–1060, Orlando, Florida, July 1999. Morgan Kaufmann.
16. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
17. R.E. Marmelstein and G.B. Lamont. Pattern classification using a hybbrid genetic program - decision tree approach. In *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann, 1998.
18. C.J. Merz and P.M. Murphy. In *UCI repository of Machine Learning*, http://www.ics.uci/mlearn/MLRepository.html, 1996.
19. N.I. Nikolaev and V. Slavov. Inductive genetic programming with decision trees. In *Proceedings of the 9th International Conference on Machine Learning*, Prague, Czech Republic, 1997.
20. J. Ross Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Mateo, Calif., 1993.
21. J. Ross Quinlan. Bagging, boosting, and c4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence AAAI96*, pages 725–730. Mit Press, 1996.
22. M.D. Ryan and V.J. Rayward-Smith. The evolution of decision trees. In *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann, 1998.
23. M. Tomassini. Parallel and distributed evolutionary algorithms: A review. In P. Neittaanmki K. Miettinen, M. Mkel and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, J. Wiley and Sons, Chichester, 1999.