A Cellular Genetic Programming Approach to Classification

Gianluigi Folino
ISI-CNR, c/o DEIS
Univ. della Calabria
87036 Rende (CS), Italy
folino@si.deis.unical.it

Clara Pizzuti
ISI-CNR, c/o DEIS
Univ. della Calabria
87036 Rende (CS), Italy
pizzuti@si.deis.unical.it

Giandomenico Spezzano
ISI-CNR, c/o DEIS
Univ. della Calabria
87036 Rende (CS), Italy
spezzano@si.deis.unical.it

Abstract

A cellular genetic programming approach to data classification is proposed. The method uses cellular automata as a framework to enable a fine-grained parallel implementation of GP through the diffusion model. The main advantages to employ the method for classification problems consist in handling large populations in reasonable times, enabling fast convergence by reducing the number of iterations and execution time, favouring the cooperation in the search for good solutions, thus improving the accuracy of the method.

1 Introduction

Data classification is a learning process that identifies common characteristics in a set of objects contained in a database and categorises them into different groups (*classes*). To build a classification a sample of the tuples (also called examples) of the database is considered as the *training set*. Each tuple is composed of the same set of attributes, or features, which are used to distinguish them, and an additional known class attribute that identifies the class that the tuple belongs to.

The task of classification is to build a description or a model for each class by using the features available in the training data. The models of each class are then applied to determine the class of the remaining data (*test set*) in the database. Data classification, originally considered a machine learning activity, plays an important role in the database field since it is considered a major *data mining* task (Fayyad 1996), which consists in the extraction of interesting and novel knowledge from real-world databases.

Decision trees (Quinlan 1993) currently represent one of the most highly developed techniques for the classification of databases. A decision tree is a tree where the leaf nodes are labelled with the classes while the non leaf nodes (decision nodes) with the attributes of the training set. The branches living a node represent a test on the values of the attribute. The path from the root to a leaf represents a set of conditions attribute-value (a rule) which describes the class labelling that leaf. There is a rule for every leaf node, thus a class is modelled by a set of rules. Decision trees are evaluated with respect to two parameters: *accuracy* and *size*. Accuracy measures the rate of misclassification. A totally accurate tree should correctly predict the class of any example from the database. The size regards the number of nodes of the tree. The simpler is the tree, the more concise is the class description and the information described can be easily understood.

C4.5 (Quinlan 1993) is the most famous decision tree based classification method. It constructs a decision tree by selecting an attribute as the root of the tree and the branches are made of all the different values this attribute can have. If all the examples at a particular node belong to the same class that node is a leaf node and it is labelled with the class, otherwise an attribute, that does not occur on the path up to the root, is chosen and branches are created for all its possible values. The algorithm terminates when all the leaves are labelled with a class. The selection of the attributes is made by using the *information gain* criterion. This criterion chooses the attribute providing the highest information gain, which is the attribute that minimises the information needed in the resulting subtree to classify the examples. The construction of simple and accurate decision trees is a demanding task. In the last few years new alternative methods to reach this objective have been proposed. Among them, *genetic programming* seems particularly suitable because of its underlying data structure.

Genetic programming (*GP*) (Koza 1992) had already been demonstrated to be able to classify a database by evolving decision trees. Koza, in fact, for a simple database showed that, after few generations, a decision tree had been generated which classified all the training cases in the right class. When applied to large dataset, however, its performance drastically degrades due to the necessity to deal with large populations of trees where each tree is constituted by a high number of nodes.

In this paper a new approach to data classification based on cellular genetic programming is presented. 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 this approach are that it can handle large populations in a reasonable time, it enables fast convergence by reducing the number of iterations and the execution time, it favours cooperation in the search for good solutions thus improving the accuracy of the method. Furthermore, it allows for the construction of a scalable genetic programming method for the classification of large databases.

The paper is organised as follows. In section 2 the standard approach to data classification through genetic programming is showed along with a brief survey of the most recent proposals. In section 3 the cellular genetic programming method is presented. In section 4 we give preliminary results obtained by a sequential implementation of the method which simulates the framework of cellular automata and shows the applicability of the approach.

2 Data Classification through Genetic Programming

Genetic programming is a variation of genetic algorithms in which the evolving individuals are themselves computer programs instead of fixed length strings from a limited alphabet of symbols (Koza 1992). Programs are represented as trees with ordered branches in which the internal nodes are *functions* and the leaves are so-called *terminals* of the problem. The search space in genetic programming is the space of all computer programs composed of functions and terminals appropriate to the problem domain. The *GP* approach evolves a population of trees by using the genetic operators of *reproduction*, *recombination* and *mutation*. Each tree represents a candidate solution to a given problem and it is associated with a *fitness value* that reflects how good it is, with respect to the other solutions in the population. The reproduction operator copies individual trees of the current population into the next generation with a probability proportionate to their fitness. The recombination operator generates two new individuals by crossing two trees at randomly chosen nodes and exchanging the subtrees. The two individuals participating in the crossover operation are again selected proportionate to fitness. The mutation operator replaces one of the nodes with a new randomly generated subtree. There is also the possibility for preventing trees to become too deep and for simplifying them.

Genetic programming can be used to inductively generate decision trees for the task of data classification. Decision trees can be interpreted as composition of functions 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. Thus there are as many functions as there are attributes. 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 outcoming from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed. The fitness is the number of training examples classified in the correct class. Both crossover and mutation must generate syntactically correct decision trees. This means that an attribute can not be repeated more than once in any path from the root to a leaf node. In order to balance the accuracy against the size of tree, the fitness is augmented with an optional parameter, the *parsimony*, which measures the complexity of the individuals. Higher is the parsimony, simpler is the tree, but accuracy diminishes.

Several methods to data classification based on genetic programming have recently been proposed. Nikolaev and Slavov (Nikolaev and Slavov 1997) present a study which proves that different fitness functions for inductive genetic programming with decision trees generate fitness landscapes with different statistical characteristics. A landscape consists of points and their fitnesses. They employ the fitness distance correlation which measure if the known optima are easily accessible from a given point on the landscape. Thus the differences among the decision trees could precisely be identified since a search point could have a small fitness value, however far away from the global optima and used for guiding the search.

Marmelstein and Lamont (Marmelstein and Lamont

1998) introduce a method for finding decision region boundaries by evolving a hybrid GP-decision classifier. A standard GP classifier (Tackett 1993) has the terminal set consisting of the database attributes and the function set of mathematical ($+$, -, $\times$, $\div$) and comparison ($\leq$) operators. The method incorporates a GP classifier into a decision tree and builds the tree by using genetic programming for the implementation of decision nodes. An m class problem is decomposed into a two-class problem: the target class and the other classes. Each node in the tree is evolved to categorise data into two groups corresponding to the target class and all the others. In this way each node classifies a smaller subset of the data considered by its parent node.

Rayan and Rayward-Smith (Ryan and Rayward-Smith 1998) present a hybrid genetic algorithm which allows genetic programming to evolve decision trees. They make use of the C4.5 algorithm to initialise the population and, each time the crossover operator is executed, leaf nodes of the generated offspring are substituted by a subtree produced by C4.5. The C4.5 algorithm is modified such that, instead of applying the information gain heuristic, it randomly selects the attributes to be tested in the functional nodes. Though the results obtained are good in terms of accuracy and size, the hybrid genetic algorithm makes use of the Quinlan's method too many times thus it fails to reduce the running time.

Freitas (Freitas 1997) proposes a genetic programming framework for classification and generalised rule induction. In his approach an individual is a tree where the terminal set consists of the attributes and their values, whether the function set consists of the logical connectives {AND,OR, NOT} and the comparison operators $\{>, \geq, <, \leq, \neq\}$. The tree is the representation of an SQL query and he suggests a tight integration between GP based classification and relational databases.

Genetic programming, thus, showed to be a particularly suitable technique to deal with the task of data classification by evolving decision trees. Interesting results, however, have been obtained when it is applied to problems that evolve small decision trees. If the database contains a high number of examples with many features, large decision trees are requested to accurately classify them. In data mining applications, databases with several millions of examples are common. A decision tree generator based on genetic programming should then cope with a population of large sized trees. Furthermore, it has already been pointed out (Ryan and Rayward-Smith 1998) that, in order to obtain the same classification accuracy of a decision tree generated by C4.5, small population size is inadequate. Processing large populations of trees that contain many nodes considerably degrades the execution time and requires an enormous amount of memory. The utilisation of parallel strategies to increase the performances of genetic programming and to realise a really scalable data classification package based on genetic programming, for data mining applications, seems the favourable approach. Parallel algorithms can deal with very large sized problems because they decompose a problem into subproblems and use parallel machines to simultaneously process the different subproblems. The exploitation of the inherent parallelism present in many classification algorithms could be the solution to obtain a scalable behaviour.

In the next section a cellular genetic programming approach for data classification is proposed. 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, favouring the cooperation in the search for good solutions, thus improving the accuracy of the method.

3 The Cellular Genetic Programming Approach

Genetic programming is well suited to parallel implementation. Parallel implementations of GP involve two main approaches. The *island* model (Martin & al.1997) and the *diffusion* model (Pettey 1997).

The island model uses a coarse grained approach. It has been adopted by Gordon (Gordon & al. 1993) to produce parallel implementations of genetic algorithms and also by Koza (Koza & al. 1995) for genetic programming. In such a model the population is subdivided into smaller subpopulations. A standard genetic programming algorithm works on each partition and is responsible for initialising, evaluating and evolving its own subpopulation. The standard GP algorithm is modified by the addition of a migration operator. Each GP process, working on a partition, is considered to be an island, detached from the other processes. Every few generations, after the fitness evaluation phase, migration occurs, whereby certain individuals are moved between processes, thus distributing the genetic material throughout the entire process 'islands'. Generally, the top 10% of individuals from each island are migrated every 10 generations.

In the diffusion model each individual has a spatial location on a low-dimensional grid and the individuals interact locally within a small neighbourhood. This model considers the population as a system of active individuals that interact only with their direct neighbours. Different neighbourhoods can be defined for the cells. The most common neighbourhoods in the two-dimensional case are the 4-neighbour (*von Neumann neighbourhood*) consisting of the North, South, East, West neighbours and 8-neighbour (*Moore neighbourhood*) consisting of the same neighbours augmented with the diagonal neighbours. Fitness evaluation is done simultaneously for all the individuals and selection, reproduction and mating take place locally within the neighbourhood. Information slowly diffuses across the grid thus clusters of solutions are formed around different optima. Recent researches proved that such an approach shows no degrade of the solution quality (Gorges-Schleuter 1992) and an improved performance.

Cellular automata (CAs) (Toffoli & al. 1986) can be used as a framework to enable a fine-grained parallel implementation of GP through the diffusion model. A CA is composed of a set of cells in a regular spatial lattice, either one-dimensional or multidimensional. Each cell can have a finite number of states. The states of all the cells are updated synchronously according to a local rule, called a *transition function*. That is, the state of a cell at a given time depends only on its own state at the previous time step and the states of its "nearby" neighbours (however defined) at that previous step. Thus the state of the entire automaton advances in discrete time steps. The global behaviour of the system is determined by the evolution of the states of all the cells as a result of multiple interactions.

A *cellular genetic programming algorithm (CGP)* can be designed by associating with each cell of a CA two substates: one contains an individual (tree) and the other its fitness. At the beginning, for each cell, a tree is randomly generated and its fitness is evaluated. Then, at each generation, every individual undergoes one of the genetic operators (reproduction, crossover, mutation) depending on the probability test. If crossover is applied, the mate of the current individual is selected as the neighbour having the best fitness and the offspring is generated. The current string is then replaced by one of the two offspring, the one having the best fitness. It is worth to notice that this replacement strategy differs from the standard approach of *cellular genetic algorithms* introduced in (Whitley 1993) where only if one of the two offspring has a better fitness than the current string, it becomes the current string.

```
Let $p_c$, $p_m$, $p_r$ be the crossover, mutation
 and reproduction probability
for each cell $i$ in CA do in parallel
  generate a random individual $t_i$
  evaluate the fitness of $t_i$
end parallel for
while not MaxNumberOfGeneration do
 for each cell $i$ in CA do in parallel
  generate a random probability $p$
   if ($p < p_c$) then
    select the cell $j$, in the neighborhood of $i$,
     such that $t_j$ has the best fitness
    $(u,v)$=crossover($t_i$,$t_j$ )
    $t_i$= best_fitness($u$,$v$)
   else
   if ($p < p_m + p_c$) then
    mutate the individual
   else
    copy the current individual in the population
   end if
   end if
 end parallel for
end while
```

Figure 1: Pseudo-code of the cellular genetic programming algorithm.

Table 1: Databases description

DATABASE	ATTR.	TUPLES	CLASSES
Cancer	9	286	2
Crx	15	690	2
Hypo	29	3772	5
Iris	4	150	4
Vote	16	435	2

The cellular genetic algorithm on a 2-dimensional toroidal grid can be described by the pseudo-code shown in figure 1, that outlines the transition function of each cell.

This approach has the advantage of working with large populations, by enabling fast convergence, and reducing the number of iterations and execution time. The cellular model avoids the problem of premature convergence in some GP applications, i.e. a rather good individual, with a fitness higher than the others, spreads rapidly through the population. In the cellular implementation of GP the produced information flows and spreads like a slow migration to the zones near the interested neighbour. So good schemata discovered can slowly diffuse through the whole population, leaving time to discover other schemata at different portions.

Table 2: Results generated by Cellular GP

	C4.5			**Cellular GP**		
DB	**Size**	**Train set**	**Test set**	**Size**	**Train set**	**Test set**
Cancer	41	19.9	30.6	11 (18.60)	21.99 (24.76)	18.95 (22.42)
Crx	44	5.9	11.7	28 (17.70)	10.61 (12.27)	14.00 (16.25)
Hypo	21	0.2	0.9	16 (18.40)	1.31 (1.62)	1.35 (1.92)
Iris	7	1.0	6.3	9 (6.50)	2.00 (2.60)	4.00 (5.00)
Vote	7	4.3	6.9	7 (7.30)	4.33 (4.63)	2.96 (2.99)

Furthermore, this approach allows to keep a diversity of the population as the search proceeds in the search space because of the non greedy technique adopted in the recombination process.

4 Implementation and Experimental Results

In this section we present the experiments and results obtained by a preliminary implementation of the method on a sequential machine. The software architecture of the environment used for the experimentation consists of three principal components: a user interface (UI), the CGP classifier, a viewer to visualise the trees. By the UI, the user can define the size of the population, the probability with which to perform reproduction, crossover and mutation and the features to be discretized. The CPG classifier has been implemented in C by modifying the *sgpc*1.1 standard tool for genetic programming (Tacket and Carmi) to meet the requirements of classification. In order to simulate the cellular automata framework, the population has been mapped into a two-dimensional array of fixed dimensions. The proportionate selection procedure has been replaced with a selection that works locally within a Moore neighbourhood. Furthermore, a procedure that does not allow the generation of trees with repeated attributes on the branches, after the application of crossover and mutation operators has been added. CGP accepts discretized data sets (training and test set) as input. If a data set contains continuos features it must first be dicretized. After the execution CGP provides a report that contains the main parameters of the simulation and, for each generation, information about the best tree, with respect to the test errors, and the standardised, average, raw and validation fitness. Furthermore, it reports the best tree with the training and test error and the size. To improve the readability of the results a viewer that visualise the best tree found has been added. It is based on the *dotty* tool available with the MLC++ library (Kohavi & al. 1994). The environment runs on a Sun Ultra-spark workstation with two 200-Mhz processors and 256 Mbytes of memory.

Experiments have been executed on five standard databases contained in the UCI Machine Learning Repository (Merz and Murphy 1996). Table 1 contains the description of these databases. They present different characteristics in the number and type (numerical and nominal) of attributes, two-classes versus multiple classes and number of examples. In particular *Crx* and *Hypo* have both nominal and numerical features. In order to deal with numeric features, they have been discretized by using the Discretize utility of MLC++ Machine Learning Library (Kohavi & al. 1994) before running the CGP algorithm. A population of 400 elements has been used with a probability of 0.095 for reproduction, 0.890 for crossover and 0.01 for mutation. The maximum depth of the new generated subtrees is 4 for the step of population initialisation, 5 for crossover and 2 for mutation. The algorithm stops after 200 generations. In table 2 the results generated by C4.5 with pruning and those of the CGP method, with a value of 0.05 for the parsimony, are presented. As already pointed out, higher values of parsimony biases the method towards smaller trees but with lower accuracy. The results have been obtained by running the algorithm 10 times and the best result with respect to the misclassification error on the test set is shown along with the average result in parenthesis. It is clear from the table that the trees generated by the CGP algorithm with respect to C4.5 are smaller, for almost all the dataset, they have a misclassification error on the training set comparable, but, more important, they generalise better than C4.5. In particular, for the cancer dataset, the results are very good. The tree contains 11 nodes with respect to 41 of C4.5 and the error test is 18.95 instead of 30.6. The quality of the outcome is encouraging, bearing also in mind that no optimisation on the size of the trees, such as editing operators or pruning, has been implemented. Future work will actually realise the parallel implementation of the method.

5 Conclusions

A new approach to data classification based on a cellular genetic programming framework has been presented. The approach showed good performance with respect to Quinlan's C4.5 method by generating both smaller and more accurate trees for almost all the databases. It has successfully been applied to standard machine learning problems. However, in order to apply the method to real data mining applications, a number of problems have to be resolved. It is known that the most expensive step of decision tree based classifiers is the fitness evaluation. The major computational effort is in fact consumed by its calculation. Though we suggested the cellular approach to be able of to cope with large populations containing big trees, we did not address the measurement of the fitness. The parallel implementation of our approach must take into account this problem if a real scalable algorithm must be realised.

References

U.M. Fayyad, G. Piatesky-Shapiro and P. Smith (1996). From Data Mining to Knowledge Discovery: an overview. In U.M. Fayyad & al. (Eds) *Advances in Knowledge Discovery and Data Mining*, pp.1-34, AAAI/MIT Press.

A.A. Freitas (1997). A Genetic Programming Framework for two Data Mining Tasks: Classification and Generalised Rule Induction. *GP'97: Proc. 2nd Annual Conference*, pp.96-101, Stanford University, CA, USA.

V. S. Gordon and D. Whitley (1993). Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest (editor), *Proceedings of the 5th International Conference on Genetic Algorithms.* Morgan Kaufman.

M. Gorges-Schleuter (1992). Comparison of local mating strategies in massively parallel genetic algorithms, in *R. Manner and B. Manderick (eds.) Parallel Problem solving from Nature II.* North Holland, pp. 554-561.

R. Kohavi,G. John, R. Long,D. Manley and K. Pfleger (1994). MLC++ a Machine Learning Library in C++. *International IEEE Conf. on Tools with Artificial Intelligence*, IEEE Computer Society Press. *http://www.sgi.com/Technology/mlc.*

J. R. Koza (1992). *Genetic Programming: On Programming Computers by Means of Natural Selection and Genetics*, MIT Press.

J. R. Koza and D.Andre (1995) Parallel genetic programming on a network of transputers. *Technical Report CS-TR-95-1542, Computer Science Department*, Stanford University.

N.I. Nikolaev and V. Slavov (1997). Inductive Genetic Programming with Decision Trees. *Proceedings of the 9th International Conference on Machine Learning*, Prague, Czech Republic, April 1997.

R.E. Marmelstein and G.B. Lamont (1998). Pattern Classification using a Hybbrid Genetic Program - Decision Tree approach. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.

W.N. Martin, J. Lienig and J. P. Cohoon (1997), Island (migration) models: evolutionary algorithms based on punctuated equilibria, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation.* IOP Publishing and Oxford University Press.

C.J. Merz and P.M. Murphy (1996). UCI repository of Machine Learning. *http://www.ics.uci/mlearn/MLRepository.html.*

C. C. Pettey (1997), Diffusion (cellular) models, in T. Back, D.B. Fogel, Z. Michalewicz (eds.), *Handbook of evolutionary Computation.* IOP Publishing and Oxford University Press.

J. Ross Quinlan (1993). *C4.5 Programs for Machine Learning.* San Mateo, Calif.: Morgan Kaufmann.

M.D. Ryan and V.J. Rayward-Smith (1998). The Evolution of Decision Trees. *Proceedings of the Third Annual Conference on Genetic Programming*, Morgan Kaufmann.

W.A. Tackett (1993). Genetic Programming for feature discovery and image discrimination. *Proceedings of the Fifth International Conference on Genetic Algorithms.*

W.A. Tackett and A. Carmi. Simple Genetic Programming in C. Available through the genetic programming archive at *ftp://ftp.io.com/pub/genetic-programming/code/sgpc1.tar.Z.*

T. Toffoli and N. Margolus (1986). *Cellular Automata Machines A New Environment for Modeling.* The MIT Press, Cambridge, Massachusetts.

D. Whitley (1993). Cellular Genetic Algorithms, *Proc. Fifth Int. Conference on Genetic Algorithms.*