Grammatical Evolution Guided by Reinforcement

Jack Mario Mingo, Ricardo Aler

Abstract—**Grammatical Evolution is an evolutionary algorithm able to develop, starting from a grammar, programs in any language. Starting from the point that individual learning can improve evolution, in this paper it is proposed an extension of Grammatical Evolution that looks at learning by reinforcement as a learning method for individuals. This way, it is possible to incorporate the Baldwinian mechanism to the evolutionary process. The effect is widened with the introduction of the Lamarck hypothesis. The system is tested in two different domains: a symbolic regression problem and an even parity Boolean function. Results show that for these domains, a system which includes learning obtains better results than a grammatical evolution basic system.**

I. INTRODUCTION

Grammatical Evolution [1], is an Evolutionary Algorithm able to develop, starting form a grammar, programs in any language. It differs from Genetic Programming [2] in that the individuals who represent the programs are not trees, but variable length linear genome. In order to generate a program from a genome, we need a mapping process that transforms any of the codons (8 bits groups) that compose the genome in the necessary information to select a BNF grammar production rules.

A grammar is used to build the program using the production rules depending on the genome, starting from the start symbol. In order to select a production rule, a codon of the genome is read, and a formula is applied; with this formula, a number which represents the production rule to be used is obtained. This way, the genome is passed through until a complete program is built. The use of a grammar guarantees the syntactical correctness of the programs.

Starting from the point that individual learning can improve evolution, [3], in this paper the intention is to demonstrate the reinforcement learning effects in population evolution. Interaction between learning and evolution was initially proposed in [4] and, it is sometimes called Baldwin effect. Following this effect, the organisms that learn, develop in a much quickest way than the ones that do not, even though when their initial genotype features do not include what has been previously learnt. In fact, the possibility that what has been previously learnt comes back to the genotype again was proposed by Lamarck [5]. Although Lamarck hypothesis is not actually accepted in biology, there is no reason for not including it in the computational view.

With these premises, a Grammatical Evolution System Guided by Reinforcement is proposed. Here, the Lamarck mechanism to substitute the original genotype with features learnt by the phenotype is also included.

Manuscript received March 30, 2007.

J. M. Mingo is with the Computer Science Department, University Carlos III of Madrid, Spain (e-mail: jmingo@inf.uc3m.es).

R. Aler is with the Computer Science Department, University Carlos III of Madrid, Spain (e-mail: aler@inf.uc3m.es).

This article has been financed by the Spanish founded research MCyT project OP:LINK, Ref:TIN2005-08818-C04-02.

II. GRAMMATICAL EVOLUTION

Grammatical Evolution [1], [6], can be considered a way of Genetic Programming that uses variable length linear genome to represent programs. These genomes usually contain binary information that has to be transformed into integer information in a process known as *transcription*. After this, the integer string is used in a mapping process, called *translation*, which consists in going throughout the genome from the left to the right in order to get the integer numbers, and apply a formula which produces as a result a value that represents the production rule to be applied. Production rules are applied starting with the start symbol of the grammar and the objective is to apply rules to each non terminal symbol not solved from the derivation tree that represents the program.

In the Grammatical Evolution initial implementation [1], the integer numbers included in the genome are in the range 0 255 and the formula to be applied is the following:

Rule = Codon Integer Value % Number of Rules

where *%* means the module operator and *Number of Rules* means the maximum number of rules to the non terminal to be expanded in each moment

In order to compare the effect of learning in the system, a standard Grammatical Evolution without reinforcement and with some specific features was initially created. In our standard Grammatical Evolution system, the range of the integer numbers is reduced to the interval 0 (*Maximum Number of Rules 1)*. This way, the binary genotype is reduced to X bits, being X the minor integer which complies 2^x *= Maximum Number of Rules*.

The Grammatical Evolution system without learning proposed is completed with a traditional generational mechanism and with the crossover, mutation, duplication and pruning operators proposed in [1]. These operators are applied in an elitist fashion.

III. AUTOMATIC PROGRAMING AND REINFORCEMENT

The possibility of widening the Genetic Programming with Reinforcement Learning [7] is an attractive topic, object of previous proposals. In [8], the possibility that in the leave nodes of the tree, actions selection funtions appear is included, leaving the internal nodes as classification elements about the status of the program. This way, the reinforcements obtained selecting actions control the fitness evaluation. This Reinforced Genetic Programming system has shown interesting results in autonomous navigation tasks. In [9], the idea of neural program is proposed, whose main difference with a genetic program is that these neural programs are a data flow structure between distributed neural processors instead of the control flow structure typically associated to genetic programs.

While the two previous approaches can be considered an automatic programming mechanism widened with a reinforcement mechanism, in [10], the opposite effect is proposed and genetic programing is applied to improve the learning by reinforcement, helping Q tables to develop.

In the Grammatical Evolution Guided by Reinforcement, we continue with the idea of extending the Evolutionary Programming with learning, but a grammar is used as the main component for building programs. These programs are control flow structures. Learning occurs by using reinforcement on the rules of the grammar used to build good programs. Grammatical Evolution Guided by Reinforcement is similar to the Learning Classifier Systems in the sense that the former also uses reinforcement learning for Q value update for rule selection. However, our system uses Reinforcement Learning to learn which productions of a grammar are more useful to create high fitness programs. Also, the systems with grammatical genetic code, as Grammatical Evolution, can induce sequentiality in uniformly scaled problems as is shown in [11]. Sequential learning requires an exponential population size as problem size increases for reliable genetic search. Grammatical Evolution Guided by Reinforcement might be an alternative to consider. However, this hypothesis should be analyzed in detail.

IV. GRAMMATICAL EVOLUTION GUIDED BY REINFORCEMENT

The Grammatical Evolution Guided by Reinforcement proposes a new way of looking at the individuals in the population. Each of them can be considered as a programmer dedicated to the task of creating a program using the grammatical rules. Certainly, this idea already appears in the Grammatical Evolution. The real novelty is in giving any of the individuals programmers the possibility of re writing their own program. This can be done if we allow each of the individuals to learn. The process is easy: initially the individual programmer is evaluated by executing the initial program of its own chromosome; then, it is allowed to create new programs. In order to do that, it will simply work with other grammatical rules put together for build new programs. In the case that any of the learnt programs is better that the original genetic program, the Lamarck hypothesis is used and the genetic program is substituted by the learnt program.

Basically, the process consists in three stages, which can be considered analogous to a biological genetic system:

- Transcription
- Translation
- Learning

Graphically, this process is shown in figure 1.

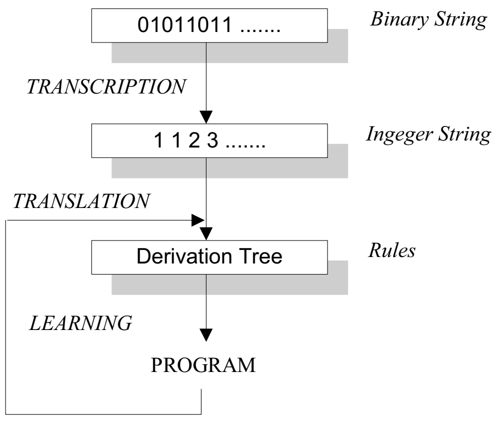

Fig. 1. Stages in the Grammatical Evolution Guided by Reinforcement.

The transcription and translation processes are basically the ones described in [6], with the features commented in section II. The individuals programmers use reinforcement learning although they could use any other way of learning. As a learning mechanism not supervised, reinforcement has the advantage that it allows the individual to verify the validity of his own actions, in this case, the appropriateness of the production rules chosen to create the program. Depending on the rewards received, the individual will create a policy that will allow it to choose the most appropriate production rule in any case.

A. Reinforcement learning

In Reinforcement Learning [7], [12] an agent is in a determined state in any moment and can chose an action among a group of possible actions. As a result of the choosing, the agent will be in a new state receiving a signal which determines the value of his selection. The agent's objective is finding an optimum policy that allows him to select the best action in each state.

Translating this situation to the Grammatical Evolution Guided by Reinforcement the following elements appear:

- An individual programmer that represents the agent.
- A group of states that represent the derivation steps used up to a point in the building of the program.

- A group of production rules applied in each of the states.

The aim of the individual programmer is to calculate a policy for the actions appropriate for any state. In the Grammatical Evolution Guided by Reinforcement, an *action* is simply a production rule to be applied. For the definition of a state, it is necessary to have into account that the process of program creation consists in applying a derivation steps sequence starting from the start symbol. It is considered that a *state* is the partial derivation steps sequence applied up to a particular instant.

This way, the 0 state would correspond to the start symbol of the grammar; state 1, to the result of substituting the start symbol by one of the production rules appropriate to it; state 2, would result from substituting the non terminal left most symbol by a production rule valid for it; and so on. This process can conclude because of two reasons:

1) The individual generates an analysis tree, that is, a tree whose leaves only contain terminal symbols. In this case, the learnt program is evaluated and its fitness is obtained.
2) The individual does not generate an analysis tree after a determined number of rules selections. In this case, the individual obtains a very poor fitness.

This process is repeated as many times as it is established by means of a system parameter, and the consequence will be the creation of a series of programs learnt by the individual. Some of these programs could be wrong, and will obtain a very poor fitness, while others, will be grammatically right and will obtain a fitness depending of their capacity to solve the problem.

B. Q Tree

As a reinforcement mechanism, the Q learning method has been followed [7], [12], [13]. Otherwise, since the state space can be very big for any domain, instead of using a table for Q values, a tree called *Q tree* is used.

The Q tree objective is to maintain the policy of actions more appropriate for each state. That is, the optimum production rules in each instant. As a result, in any node of the Q tree the following information is maintained:

- A numeric value which represents the production rule used to get to the state
- A group of numeric values which represents the different production rules that can be applied in the state

A simple example will be useful to clarify the concepts and show the Q tree's content. Suppose a BNF grammar composed by the tuple [N, T, S, P] that includes the following elements:

N = {<expr>, <op>, <var>}
*T = {+, , *, /, X}*
S = {<expr>}

And *P* is composed by the production rules:

```
<expr> ::= <expr> <op> <expr>(0)
         | <var>               (1)
<op> ::=  +   (0)
         |    (1)
         | *  (2)
         | /  (3)
<var> ::= X   (0)
```

For a single chromosome like the following integer string resulting from applying the transcription process to the initial binary string:

TABLE I
CHROMOSOME OF AN INDIVIDUAL PROGRAMMER

0	3	1	2	1	2	1	0

The mapping process builds the analysis tree:

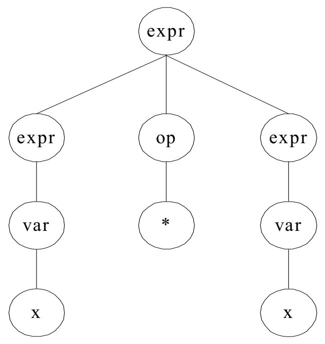

Fig. 2. Derivation tree for the single chromosome.

Resulting from applying the known formula to the non terminal symbols:

$$Rule = Integer\ Value\ \%\ Number\ of\ Rules$$

In the derivation tree, the internal nodes correspond to the non terminal symbols and the leaves correspond to the terminal symbols.

We have to highlight that in the mapping process of this single individual only the first 6 genes are used. The remaining genes are introns. To reduce the introns, the operator prune was introduced in [1]. Originally, the prunning has been applied with a probability to any individual that does not map all of its genes. In our approach, the prunning is applied to all individuals that do not map all their genes, deterministically. Genes not used are discarded and they do not take part in the genetic operations, with the exception of the mutation operator. Theoretically, the prunning increases the likelihood of beneficial crossovers, avoiding selecting crossover points which do not take part in the mapping process.

In parallel to the building of the derivation tree the Q tree is built, which in the case of the individual of the example corresponds to:

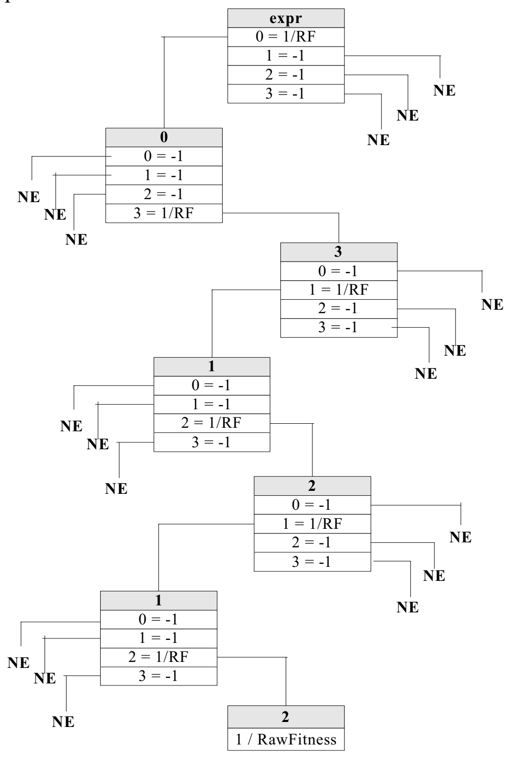

Fig. 3. Q tree for the individual chromosome.

where *RF* makes reference to *Raw Fitness* [2], obtained by the individual and *NE* means "*Not Expanded*" to indicate that the branch has not been explored yet.

Initially, each entry of the Q tree is configured with the default value 1. In each node there are as many entries as the maximum number of production rules (in the example the maximum would be 4 rules for the non terminal <*op*>). Of course, there will be entries that will not be expanded because they are not valid for the non terminal current.

The meaning of the tree is simple. The numeric value shown in the shadowed box indicates the production rule applied, as a consequence of which we come to the state. The exception to this rule is in the initial node, which is labelled with the start symbol of the grammar. The four numeric values under the shadowed box indicate the possible production rules to be applied. This way, the node labelled with the start symbol can be expanded with four rules (though for the start symbol of the example, only two rules are possible), and in the example, the 0 rule has been chosen (the entry with "0" is equal to 1/RF). Selection of rules is determined on each step according to the exploration/exploitation tradeoff. In our system, an e greedy strategy is implemented. The other three possible rules in the initial node are labeled as "*Not Expanded*" to indicate that they have not been explored. The choosing of the 0 rule implies to go a level forward, both in the derivation tree (figure 2), as in the Q tree (figure 3). The start symbol is substituted by <*expr*><*op*><*expr*>. Since there are three non terminal symbols, the mapping process continues. Now, we must choose a new rule for the leftmost non terminal (<expr>). In the example, rule 3 for <expr> is chosen. As <expr> only has two rules, the formula "*Rule = Integer Value % Nº Max of Rules for Non Terminal*" returns the 1 value (3 % 2) and the second rule is chosen (<expr> = <var>). In the figure 3, the entry 3 for the "zero" node is equal to 1/RF. Now, both the derivation tree and the Q tree, expand a level again. Concretely, in the Q tree, the node of level two appears labelled as "three". In the derivation tree, the situation would correspond now to: <*var*><*op*><*expr*>. Again, there are non terminals symbols and the process continues by choosing the rules 1, 2, 1 and 2, which produce the *X* * *X* program. Now, the Q tree corresponds exactly to figure 3. We have to highlight that the Q tree reflects the followed path, i.e. the rules that the agent has chosen.

C. Rewards

The type of task focused by the individual programmer fits in the episodic model [7], [12] in that we start from an initial state and we let the agent to try different actions till it gets to an objective state. This process is repeated during a series of episodes.

In the Grammatical Evolution Guided by Reinforcement, the initial state corresponds to the one that comes from the start symbol and the final state is that which generates a grammatically right program. Each of the episodes corresponds to a learning step and the number of learning steps is established by means of a parameter of the system. The episode begins by the start symbol and applies the production rules till all the symbols are terminal ones. The following rewards can be established during the episode:

- If when applying a production rule, non terminal symbols still exist, the individual obtains a 0 as a reward and the mapping process continues.
- If when applying the last production rule, all the symbols are terminal ones, the generated program is executed and the raw fitness value is obtained as a reward.

This way, a useful reward is only obtained when a complete program is generated. Since the episodes must be finite, by means of a system parameter we can establish the limit of trials needed to generate a complete program. If the limit is reached and non terminal symbols still exist, the individual obtains a very poor fitness and very bad reward.

With the obtained reward, the Q value is updated in the correspondent node of the tree. As an updating rule, the following formula is used:

```
For j=leave node to root node do
        $Q(s_j, a_j) = r_j + \gamma \max_{a'} Q(s_{j+1}, a')$
```

where, r_j is the reward in the level j, s_j refers to the state in the level j of the Q tree, s_{j+1} to the state in the level $j+1$ of the Q tree, a_j is the production rule taken in the node s_j and a' is the rule with the higher value Q for node s_j.

Therefore, the update is produced from the leave node of the Q tree. This node is labelled with the last production rule, that is, with that which ends with a grammatically right program. From this node, the raw fitness value is spread to all the nodes of the derivation chain till arriving to the root. The discontinuity factor is γ and, in the system, a 1 has been used as a value, reason why in the Q Tree the value of the leave node, *1/RF,* is spread without discount to the internal nodes. This discount value can be used always under the conditions of episodic and finite tasks [7]. A discount factor equal to 1 implies that all the nodes in the path have the same value, i.e. they have equal weight. This is appropriate in cases as this, where all the nodes contribute equally to the final solution.

D. Balance between exploration and exploitation

One of the problems when the reinforcement learning is used is in the balance between the exploration of the new actions and the exploitation of the learned actions. From all the possible balancing strategies in the Grammatical Evolution Guided by Reinforcement, we have followed an e greedy strategy [7], [12].

In order to solve the randomness of this strategy, we have chosen not to use a constant e greedy factor but one that varies depending on the learning step, following the formula:

$$EGreedyFactor = 1 \quad (lerningStep / lerningStepsNum)$$

where *lerningStep* is the present learning step and *learningStepsNum* is the total number of learning steps fixed for the system (global parameter).

By means of this strategy it is possible to contribute to the exploration in the first learnings and the exploitation in the last learning steps. This strategy tries to use a number of learning steps that ensures an early almost full evaluation. This way, the last learning steps can take advantage of a Q tree with more information about the production rules tested.

E. Lamark effect

The final result of the learning process of an individual programmer will be a series of programmes, some of them right and the others wrong. The right programs will be evaluated and will produce their appropriate fitness, which can be either better or worse than the original genetic program. From the computational point of view, it is possible to modify the genetic code of the individual for him to assume the best learnt program features. This is what the Grammatical Evolution Guided by Reinforcement does; the best learnt program becomes, after a translation and transcription inverse process, a new binary genome of the individual.

V. Results

In order to test the performance of our system, it has been applied to two domains very well known in Evolutionary Computation. We are talking about the symbolic regression problem and the even 3 parity Boolean function.

In order to compare the results more precisely, in each of the domains, groups of 100 executions were done. Each execution group is done for three different systems:

- A Grammatical Evolution system without reinforcement like the one described in section II.
- A Grammatical Evolution system Guided by Reinforcement that uses a Q tree for each of the individuals in the population.
- A Grammatical Evolution system Guided by Reinforcement that uses one single Q tree for all the population.

Below there is a summary of the problems as well as the results obtained, explained more in depth. The operators used are the one point crossover [6], the mutation and the duplication [1], [6], and the elitist reproduction. The crossover is done depending on the fitness, while the mutation and the duplication are also applied in an elitist manner. All the systems follow the generational model.

A. Symbolic regression

The symbolic regression problem tries to obtain a symbolic mathematical expression from a set of input output pairs. The function to be regressed is:

$$f(X) = X^4 + X^3 + X^2 + X$$

The grammar used to solve the problem is similar to [1]:

```
N = {<expr>, <op>, <pre op>, <var>}
T = {+,  , *, /, sin, cos, exp, log, X, (, )}
S = {<expr>}

<expr> ::= <expr> <op> <expr>        (0)
         | ( <expr> <op> <expr> )    (1)
         | <pre op> ( <expr> )       (2)
         | <var>                     (3)
<op> ::=  +     (0)
       |        (1)
       |  *     (2)
       |  /     (3)
<pre op> ::=  sin    (0)
           |  cos    (1)
           |  exp    (2)
           |  log    (3)
<var> ::= X   (0)
```

Table II summarizes the configuration of the symbolic regression problem. This configuration is common to the three systems that have been tested. Afterwards, the specific parameters for each of the systems will be commented.

TABLE II
GENERAL PARAMETERS FOR THE SYMBOLIC REGRESSION PROBLEM

Objective	Finding a function of an independent variable and another one dependent in a symbolic manner which fits a 20 points sample (x_i, y_i) obtained from the fourth degree polynomial: $f(X)=X^4+X^3+X^2+X$
Fitness cases	20 pairs of points in the interval [-1,+1]
Raw Fitness	The sum of the errors for the 20 fitness cases
Standardized Fitness	Equal to the raw fitness
Termination criteria	The number of fitness cases where the error is below 0.01

1) Grammatical Evolution without reinforcement: When the problem is solved with a standard evolution system, that is, without reinforcement, apart from the general parameters, the specific parameters shown in table III are used.

TABLE III
SPECIFIC PARAMETERS OF THE GRAMMATICAL EVOLUTION WITHOUT REINFORCEMENT FOR THE SYMBOLIC REGRESSION PROBLEM

Evolutionary parameters	C 10; W 10; M 500; G 51; Pc 0.9; Pd 0.01; Pm 0.01

Where *C* means the number of codons per individual, *W* is the number of wrapping events [6], *M* is the number of individuals, *G* the number of generations and *Pc, Pd* and *Pm* the crossover, duplication and mutation probabilities. Figure 4 shows the accumulated frequency achieved by the system.

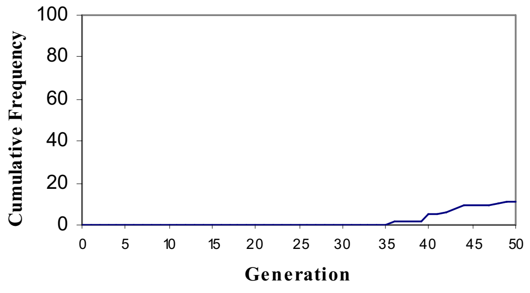

Fig. 4. Accumulated frequency for the symbolic regression without reinforcement.

The grammatical evolution system without reinforcement only gets 11% success probability for this problem. This result is similar to the original Grammatical Evolution implementation [1] when a generational mechanism is used as replacement mechanism. A steady state mechanism may obtain better results in this problem [14], but we have only compared to the generational mechanism used in the original Grammatical Evolution.

2) Grammatical Evolution Guided by Reinforcement with a Q tree for every individual: This system is distinguished by the fact that it uses a Q tree for every individual in the population. The individual Q trees are not shared by individuals of the population. Moreover, these individual Q trees are removed after every generation, because all the individuals are renewed, unless the individual is selected in an elitist reproduction. Table IV summarizes the specific features of this system.

TABLE IV
SPECIFIC PARAMETERS OF THE GRAMMATICAL EVOLUTION GUIDED BY REINFORCEMENT AND A Q TREE PER INDIVIDUAL FOR THE SYMBOLIC REGRESSION PROBLEM

Evolutionary parameters	C 10; W 10; M 300; G 51; Pc 0.9; Pd 0.01; Pm 0.01
Learning parameters	L 300; γ 1

where L refers to the number of learning steps and γ is the discount factor. Figure 5 shows the accumulated frequency achieved by the system:

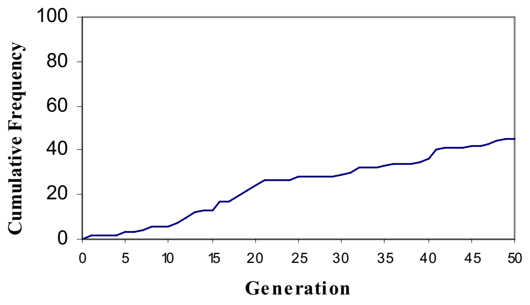

Fig. 5. Accumulated frequency for the symbolic regression with reinforcement and *n* Q trees.

The introduction of reinforcement means an important improvement going from 11% to 45% success probability.

3) Grammatical Evolution Guided by Reinforcement with a Q tree for the whole population: The only difference between this system and the one stated before is that now we only use a single Q tree for all the population. With this approach the global knowledge, obtained through reinforcement learning, is shared by all individuals. In each of the learning steps, each individual explores or exploits a branch of the single Q tree. A main difference with the previous system is that the single Q tree is kept between generations. Table V summarizes the specific parameters of the system.

TABLE V
SPECIFIC PARAMENTERS OF THE GRAMMATICAL EVOLUTION GUIDED BY REINFORCEMENT WITH ONE SINGLE Q TREE FOR ALL THE POPULATION FOR THE SYMBOLIC REGRESSION PROBLEM

Evolutionary parameters	C 10; W 10; M 300; G 36; Pc 0.9; Pd 0.01; Pm 0.01
Learning parameters	L 300; γ 1

Figure 6 shows the accumulated frequency in this case.

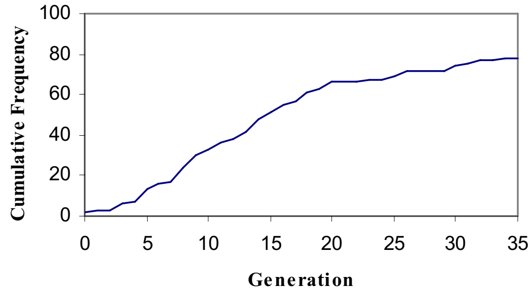

Fig. 6. Accumulated frequency for the symbolic regression with reinforcement and *1* Q Tree.

As it can be seen in the figure, the effect of the fusion of Q trees in one single Q tree is important for solving of the problem since now the success probability is 78%.

B. Even 3 parity Boolean Function

In order to solve the even 3 parity problem, the following grammar is used:

N = {<expr>, <op>, <var>}
T = {and, or, nand, nor, (,)}
S = {<expr>}

```
<expr> ::= <op> ( <expr> , <expr> )  (0)
         | <var>                     (1)
<op> ::=  and    (0)
         | or    (1)
         | nand  (2)
         | nor   (3)
<var> ::= d0     (0)
         | d1    (1)
         | d2    (2)
```

The group of terminals has been taken from [2]. Table VI shows the common configuration for the problem.

TABLE VI
GENERAL PARAMETERS FOR THE EVEN 3 PARITY BOOLEAN PROBLEM

Objective	Finding a logical function which solves the even 3 parity Boolean problem, that is, the one that returns *true* when there is an even number of bits set to 1 arguments and *false* when not
Fitness cases	The 8 possible logical combinations for three arguments
Raw Fitness	The number of success in the 8 fitness cases
Standardized Fitness	8 Raw fitness
Termination criteria	When the standardized fitness is 0

The parity Boolean functions both even or not even are very complex problems with a difficult solution with genetic methods, to the point that, for instance, the even 5 parity Boolean function is not solved with standard Genetic Programming [2] but the use of automatically defined functions is required [15]. In [16], a system that considers whether it is a good idea to transfer immediately improvements found by a single individual to other individuals in the population is proposed. In this system the computational effort is reduced, compared to Koza's ADFs.

1) Grammatical Evolution without Reinforcement: Table VII shows the specific parameters for the Grammatical Evolution system without Reinforcement.

TABLE VII
SPECIFIC PARAMETERS OF THE GRAMMATICAL EVOLUTION WITHOUT REINFORCEMENT FOR THE EVEN 3 PARITY BOOLEAN PROBLEM

Evolutionary parameters	C 10; W 10; M 2000; G 51; Pc 0.9; Pd 0.01; Pm 0.01

The Grammatical Evolution system without Reinforcement was not able to find a solution for this problem with 2000 individuals and 50 generations. This is not a surprising result since, for instance, in [2], 4000 individuals are used to solve the problem.

2) Grammatical Evolution Guided by Reinforcement with a Q tree for each single individual: Table VIII summarizes the specific characteristics for this system.

TABLE VIII
SPECIFIC PARAMETERS OF THE GRAMMATICAL EVOLUTION GUIDED BY REINFORCEMENT AND A Q TREE PER INDIVIDUAL FOR THE EVEN 3 PARITY BOOLEAN PROBLEM

Evolutionary parameters	C 10; W 10; M 500; G 51; Pc 0.9; Pd 0.01; Pm 0.01
Learning parameters	L 50; γ 1

This system results are shown in figure 7.

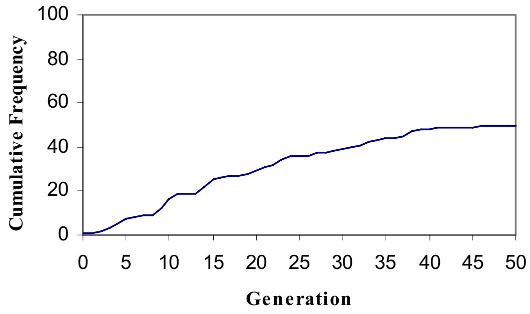

Fig. 7. Accumulated frequency of the even 3 parity Boolean function with reinforcement and *n* Q trees.

The introduction of reinforcement lets us solve this problem with a success probability of 50% if we use a Q tree per individual.

3) Grammatical Evolution Guided by Reinforcement with a Q tree for the whole population: The specific parameters of the system are shown in table IX.

TABLE IX
SPECIFIC PARAMETERS OF THE GRAMMATICAL EVOLUTION GUIDED BY REINFORCEMENT AND A Q TREE FOR THE WHOLE POPULATION FOR THE EVEN 3 PARITY BOOLEAN PROBLEM

Evolutionary parameters	C 10; W 10; M 500; G 51; Pc 0.9; Pd 0.01; Pm 0.01
Learning parameters	L 50; γ 1

Finally, figure 8 shows the accumulated frequency results for this system.

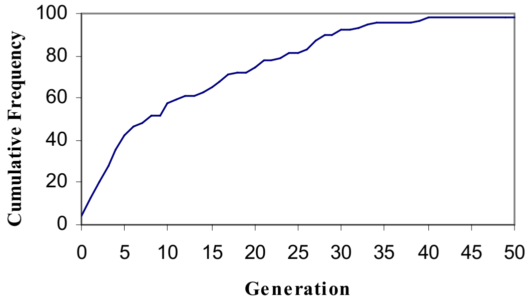

Fig. 8. Accumulated frequency of the even 3 parity Boolean function with reinforcement and one single Q tree.

The reinforcement with one single Q tree increases even more the success probability to 98%.

VI. Conclusions

A grammatical evolution system widened with reinforcement learning has been proposed in order to demonstrate one more time the beneficial effect of learning in evolution. These two mechanisms propose two ways of adaptation to the environment. From the computational point of view, these two theories correspond to two different types of searching. On one hand, the evolution acts on the population and carries out a global search in the space of possible solutions to the problem. On the other hand, the learning acts on the individual and represents a local search among the near solutions to a particular individual.

The interaction between evolution and learning produces positive results in both studied domains, although the reinforcement mechanisms influence them very much. Thus, on one side, the use of a Q tree for each of the individuals in the population improves the system without reinforcement results but does not reach more than 50% success probabilities in any of the domains. On the other side, the use of only one Q tree for the whole population offers much better results. In the symbolic regression problem, a 78% success probability is obtained, and a 98% in the case of the Boolean function.

From the analysis of these results we can conclude that keeping only one Q tree seems to be more appropriate than keeping several of them, which appears logical since only one Q tree supports the learning acquired by all the individuals; whereas multiple Q trees only support individual learning. With a single Q tree, all the individuals can use the best rules in each of the states, exploiting branches of the tree already explored by other individuals. In the case of multiple Q trees, each of the individuals can only exploit what has been already explored by itself. The counterpart of the use of one Q tree obviously is in the huge quantity of memory resources that can be needed in complex environments. This problem has been partially solved cutting off those Q tree branches that do not generate valid programs, but the problem of scalability should be examined in subsequent research.

References

[1] J. J. Collins, C. Ryan, M. O'Neill. "Grammatical Evolution: Evolving Programs for an Arbitrary Language". *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*. Springer-Verlag, 1998, pp. 83-95.

[2] J. Koza. *Genetic Programming: On the Programming of computers by Means of Natural Selection* .MIT Press, 1992.

[3] G. E. Hinton, S. J. Nowlan. "How Learning can guide evolution". *Complex Systems,* 1, 1987*,* pp. 495-502.

[4] M. J. Baldwin. "A new factor in evolution". *The American Naturalist*, 30, 1896, pp. 441-451.

[5] J. B. Lamarck. "Of the influence of the environment on the activities and habits of these living bodies in modifying their organization and structure". *Zoological Philosophy*. MacMillan, London, 1914, pp. 106-127.

[6] C. Ryan, M. O'Neill. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language.* Kluwer Academic Publishers, 2003.

[7] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[8] K. L. Downing. "Adaptative Genetic Programs via Reinforcement Learning". *Proceedings of the 3rd Genetic and Evolutionary Computation Conference.* San Francisco, California, 2001, pp. 19-26.

[9] A. Teller. "The internal reinforcement of evolving algorithms". *Advances in Genetic Programming,* 3. MIT Press, 1999, pp. 325-354.

[10] H. Iba. "Multiagent reinforcement learning with genetic programming". *Proceedings of the Third Annual Conference*. Morgan-Kaufmann, 1998, pp. 167-172.

[11] K. Ohnishi, K. Sastry, Y-P. Chen, D. E. Goldberg. "Inducing Sequentiality Using Grammatical Genetic Codes". *Proceedings of the Genetic and Evolutionary Computation Conference.* Seattle, Washington, 2004, pp. 48-59.

[12] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[13] C. Watkins. P. Dayan. "Q-Learning". *Machine Learning*, 8, 1992, pp. 279-292.

[14] M. O'Neill, C. Ryan. Grammatical Evolution: "A Steady State Approach". *Proceedings of the Second International Workshop on Frontiers in Evolutionary Algorithms.* 1998, pp. 419-423.

[15] J. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[16] R. Aler, D. Camacho, A. Moscardini. "The Effects of Transfer of Global Improvements in Genetic Programming". *Computing and Informatics (formerly Computers and Artificial Intelligence)*. Ed. By Slovak Academy Sciences Institute of Informatics, vol. 23, nº 4, pp. 374-394, Nov. 2004