Distance Modulation Competitive Coevolution method to find initial configuration independent cellular automata rules

A. Berlanga & P. Isasi & A. Sanchis & J.M. Molina
Departamento de Informática
Universidad Carlos III de Madrid
28911, Leganés, Madrid
E-mail: aberlan@ia.uc3m.es

Abstract

One of the main problems in machine learning methods based on examples is the over-adaptation. This problem supposes the exact adaptation to the training examples loosing the capability of generalization. A solution of these problems arises in using large sets of examples. In most of the problems, to achieve generalized solutions, almost infinity examples sets are needed. This make the method useless in practice.

In this paper, one way to overcome this problem is proposed,based on biological competitive evolution ideas. The evolution is produced as a result of a competition between sets of solutions and sets of examples, trying to beat one each other. This mechanism allows the generation of generalized solutions using short example sets.

1. Introduction

Coevolution is referred to the simultaneous evolution of two or more species, where the survival of each specie depends one each other. When talking about coevolution in computational terms, it is referring to the ability of a system to improve its performance by means of mutual adaptation of its different constituents. The final performance of the system is improved as a consequence of the incremental adaptation among constituents.

These ideas come from the field of biology where important theoretical studies [1] and definitions [2] have been done. The focus of these early ideas was that competition between species could guide the adaptation of each one separately. These biological ideas have been expanded in further works [3] where the competition is mentioned as the principal engine in the evolution of species, in such a way that a scaling competition takes place, yielding in an incremental and reciprocal adaptation. This kind of adaptation is known with the name of evolutionary *"arms race"* because of its analogy with the growing armament in countries, as a consequence of trying to overcome others armament.

The coevolutive process has proven its capabilities of generating complexity in nature. From the computational point of view, the arguments are quite the same. The computational system is composed by several confronted subsystems, for instance in a game (the computational system) the faced players are the confronted subsystem. The evaluation of a subsystem depends on the performance over the opposite one. There are many works that try to introduce the ideas of coevolution into the field of evolutionary computation. One of the first authors in applying the coevolution in an optimization problem was Hillis with his work over the coevolution of parasites for improving solutions in a sorting networks problem [4]. In this work, the competition takes place between individuals solution and individual examples. These examples were generated to be a good test bed for the solutions.

More recently, some works for establishing the theoretical basis of coevolution have been done. Paredis, in 1996, proposes a general framework for the use of coevolution to boost the performance of genetic search [5], and introducing a new type of Genetic Algorithm called Coevolutionary Genetic Algorithm. Rosin and Bellew [6] introduced three new techniques in competitive coevolution: Competitive fitness sharing, shared sampling, and hall of fame; and provide several different motivations for these methods and mathematical insights into their use.

Coevolution has also been used to select an optimal set of examples for problems, by means of evolution of examples. The selection of good examples is a typical problem in learning from examples techniques, mainly when dealing with complex and real problems. This is known as the *testing problem.* A perfect solution to the testing problem has to select a minimal and complete set of extremely difficult test cases, for each one of the possible solutions. Note that each solution could need a different set of examples. These sets are very difficult

0-7803-5731-0/99$10.00 ©1999 IEEE

to identify for realistic problems.

In this article,a new method is introduced, based on Hillis' ideas to use coevolution for overcoming the testing problem. A general framework to apply coevolution to any evolutionary computation technique is propodes, in such a way that examples could evolve towards optimal data examples.

2. The Testing Problem

When learning from examples, for testing a solution is necessary to face it with different situations (examples set). The problem arises when selecting examples, because a different selection of situations could ends in a very different evaluation of the solution. In other words, the evaluation of a solution is very sensitive to the selection of situations to test that solution.

To overcome this problem, Rosing and Bellew [6] suggested a new coevolutionary method, the *shared sampling*. In this method, a population of examples is always kept. Each example of the population is evaluated, computing its performances over a previously selected set of solutions. In the same way, each solution is evaluated computing its performances over a previously selected set of examples. The selection of the examples is carried out proportionally to the evaluation of the examples. Examples with better general evaluations are preferred as test cases for the solutions, and theirs evaluations computed again.

The Hillis' solution [4] is similar. In this case the examples are not selected, each time an evaluation of a solution is needed. By the opposite, each solution has a subset (subpopulation) of examples related with it. This subset is kept constant, and it is in continuous evolution.

The evolution of examples, in both of the cases, tries always to generate harder examples for the solutions. As solutions are more complex and accurate, they must prove their capabilities with more sophisticated and complex examples.

In some cases, the above mentioned solutions could generate a serious problem. Considering, for instance, one problem where the generation of good solutions over a reduced examples set is a very difficult task:

- By one hand, if examples evolve towards hard data sets, the process could ends into an impossibility of achieving solutions to the problem, and the continuous adaptation of the examples could stop the adaptation of the solutions.
- On the other hand, if the process of adaptation of examples is carried out in such a way that the adaptation of solutions is allowed, the solutions could be reached in a process of over-adaptation, difficulting the generation of more accurated solutions.

These two problems imply some difficulties in the process of evaluation of the solutions:

- Firstly, as the examples set is shared for all solutions, even if the evolution of examples was slowed down, the solutions could be faced with much more difficult examples. A good example for a solution could be a bad one for another. When examples are too difficult, there is no selective pressure for solutions to evolve. Fitness values of all solutions are similar, low fitness, and there is no way of selecting good individuals to improve solutions.
- Secondly, when a population of solutions has been adapted to an examples set, the change of the examples set could has negative consequences. The fitness landscape changes abruptly and the previously evolved solutions has no meaning of evolving towards the new fitness landscape.

For overcoming these problems, in this work a new method of adjusting coevolution is proposed, to allow: the evolution of good solutions and hard test examples in complex generalization problems. In this way, a general framework, called *Uniform Coevolution*is introduced. Independently of the particular evolutionary computation method, for implementing coevolutive ideas in problems where a complete data examples are needed and not available. The evolution of both, solutions and examples will allow the generation of shorter and harder data examples gradually, and avoiding the over-adaptation problem.

Our proposal introduce the following ideas:

Independent examples sets Each solution has an independent examples set in which computes its fitness, instead of having an examples set for all population of solutions. Therefore, examples sets evolve independently with the accuracy of its related solution.

Smoothing fitness landscape In each step of adaptation to an examples set, the individual has to deal with a new optimization problem. In many cases this is more difficult because the individual is not a randomly generated one, but a previously learned one. So, the adaptation to new examples could be extremely complicated. The smoothing fitness landscape mechanism tries to solve this problem by means of weighting the importance of the individual performance over each component in the examples set. As the solution has to deal with each example in its related examples set, it will have a different fitness value for each example. The final fitness of the solution will have to be in mind all the values computed. Usually, the average of the values is used as final fitness function. In the smoothing fitness mechanism a weighted average is used.

Examples involution In spite of all these mechanisms of adaptation, the final process could be unreachable. The continuous adaptation of the examples

could modify the fitness landscape of an individual too abruptly. The examples involution avoids this problem. In evolutionary computation methods, the evolution is always towards better individuals, in terms of fitness function. However, in examples involution mechanism, the adaptation goes both ways, towards better and worse examples. This means that examples generally evolve towards better examples, more difficult for the solution. But when the examples set is becoming so difficult as it is impossible for the solution to solve them, an evolution towards worse examples is allowed. In this way, the solutions evolve more gradually in order to solve, in the future, the previous difficult examples. This mechanism allows to overcome local optima problems, in which solutions could not evolve because the examples set is too much difficult.

3. Uniform Coevolution

In Uniform Coevolution method there are also two evolutionary systems competing. The first is related with the evolution of solutions and the last one with the evolution of examples sets for the first system.

In the figure 1 the general architecture of the Uniform Coevolution is shown. This structure reflects what it was called *Independent examples sets.* The architecture is composed of: A population of solutions and a set of populations of examples (One population of examples for each individual in the population of solutions).

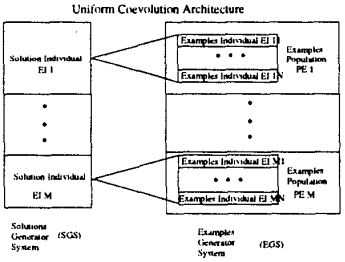

Figure 1: Architecture of the Uniform Coevolution.

The solutions and examples systems are named respectively:

- **Solutions Generator System (SGS).-** It is composed of a population of solution individuals (SI). For computing their fitness, its necessary to face each individual with a set of different situations, examples, represented by a population in the Example Generator System.
- **Example Generator System (EGS).-** It is a meta-population composed by meta-individuals, that are populations of examples (PE). Each PE is related with a SI. A PE is composed by examples individuals (EI). The fitness of those individuals is inversely proportional to its related SI's fitness, when operating over them.

The evolution of each system depends on the other evolution. The general procedure is as follows:

1. *Initialization of populations:*
   1. *SGS initialization (M SI individuals)*
   2. *EGS initialization (M PE of N EI individuals each)*
2. *Computation of the fitness*
   1. *Evaluation of SI over each individual EI in its related PE*
   2. *The fitness of SI is a combination of the above evaluations*
   3. *The fitness of PE is set inversely to the fitness value of the corresponding SI*
3. *Generation of new populations*
   1. *PE evolution by means of generation of new EI's applying an ad-hoc genetic operator (Incremental Genetic Operator -IGO-)*
   2. *EGS and SGS evolution*

3.1. Solution Generator System

The objective of this system is to gradually generate better solutions to a particular problem. Any evolutionary computation method can be used, where an individual represents one problem solution. The evolution of the SGS follows the dynamics of the evolutive computation method selected.

Computation of solution's fitness.- The generation of better solutions is driven by the fitness function. Each individual is evaluated over a set of examples. Lets call PE_i the examples set of the individual i, this population is composed of several independent blocks ($A..Z$) which meaning will be explained later. Therefore EI_{ij}^{A} is the j-th example of the block A for the set PE_i. For *smoothing fitness landscape* purposes, a linear combination of evaluations is used as fitness value of the individual. The fitness for an individual i is computed using the evaluation values of that individual over a set of N examples, in the following way:

$$F_i(t) = \Sigma_{k=A}^{Z}\Sigma_{j=1}^{N}\alpha_{ij}^{k}(t) \times f(EI_{ij}^{k}) \quad (1)$$

Where $f(EI_{ij}^{k})$ is the value of the evaluation of individual i over the example j (E_{ij}^{k}) in the block k, from its related EI's population ($\{PE_i\}$). α_{ij}^{k} values are used to weight the importance of each example in the total computation of the fitness of an individual. As the reward of the performances depends on the degree of adaptation of the solutions, α values are variable with time. The equations that drive the change of α values for one individual i, independently to the belonging block, are:

$$\alpha_{ij} = \frac{(e^{x_i}-1)(e^{y_{ij}}-1)+(e^{1-x_i}-1)(e^{1-y_{ij}}-1)}{(e-1)^2} \quad (2)$$

where x is a measure of the evolution degree of the individual over its examples set. If the individual i performs well over its related examples set, the x_i will have a value near zero:

$$x_i = 1 - \frac{\overline{f_i}}{f_{MAX}} \quad (3)$$

where $\overline{f_i}$ is the average fitness for individual i over all individuals in the same block in $\{PE_i\}$. f_{MAX} is the maximum fitness value that could ever be reached, the optimal fitness.

$$\overline{f_i} = \frac{\Sigma_{j=0}^{N} f(EI_{ij})}{N} \quad (4)$$

By the other hand, the y_{ij} value, described in equation 5, measures the importance of an example evaluation compared with others. Values of y_{ij} equal to zero means that $f(EI_{ij})$ is the lower fitness value over all individuals in the same block in $\{PE_i\}$. By the opposite, values of y_{ij} equal to one, means that $f(EI_{ij})$ is the highest fitness value over all all individuals in the same block in $\{PE_i\}$.

$$y_{ij} = \frac{f_{ij} - f_{i,min}}{f_{i,max} - f_{i,min}} \quad (5)$$

where f_{max} is the maximal fitness for individual i over the set of examples and f_{min} is the minimal fitness for individual i over the set of examples.

$$f_{i,max} = Max\{f(EI_{ij})\}_1^N \quad (6)$$

$$f_{i,min} = Min\{f(EI_{ij})\}_1^N \quad (7)$$

In figure 2 the different values of α_{ij} depending on x_i and y_{ij} can be seen. It can be seen α values for different evolution periods. In the initial situation individuals are randomly generated and their fitness values are very far from optimal ($x_i \simeq 0$). In this case the weighter evaluation corresponds with the better evaluation, this smoothing fitness mechanism, as previously mentioned, allows the evolution and improvement of slightly good individuals. As the individual adapts the difference between good and bad examples has less importance. At the end of the evolution is better to increase the weight of bad examples, in order to find a generalized solution.

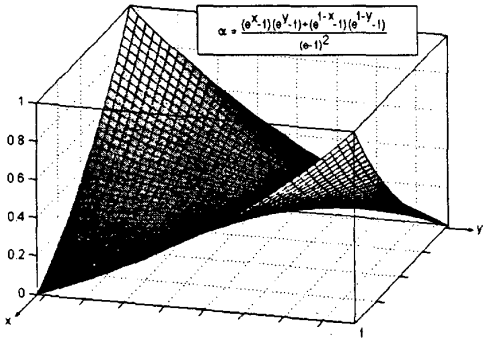

Figure 2: α_{ij} values depending on x_i and y_{ij}.

3.2. Example Generator System

A model, for being considered competitive, has to be composed of, at least, two independent systems. In Uniform Coevolution, the system faced against the SGS is called the EGS. The EGS is a meta-population composed of a set of populations of examples $\{PE_i\}$. Therefore, the EGS is composed of two dependent evolutive systems: the meta-population and the PE_i, one embedded into the other.

Meta-population.- Examples are divided in M independent sets PE_i, that are the individuals in the meta-population. Each PE_i is related with, and competes against a unique solution SI_i. Individuals PE_i are composed of a number of chromosomes. These chromosomes are the previously mentioned blocks of the PE_i. Each chromosome represents a set of examples. These blocks, that could be considered independent and evolve in an independent way, are needed for crossover purposes. When the individuals in the meta-population interchange theirs genetic materials, the blocks are interchanged (Figure 3). The meta-population evolves following the guidelines of the evolutive computation method selected for implementing the meta-population.

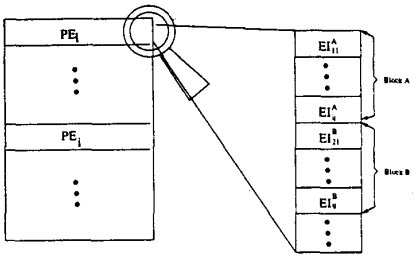

Figure 3: Composition of the meta-population.

Population PE_i.- All individuals EI_{ij} in a block are generated from an especial individual called *"seed example"*. The generation of EI_{ij} is based on a particularly designed *Genetic Operator called Incremental Genetic Operator IGO*. This generation process constitutes the only method for evolving PE_i.

Incremental Genetic Operator.- For designing the IGO, it is necessary to define a distance function between examples. This distance is a measure of the existing differences between examples, most different are two examples a higher value outputs the function and vice-versa:

$$D : E \times E \rightarrow \Re \quad (8)$$

where E is the set of all possibles examples for a particular problem. This distance measure is used for generating the individuals in a block. The "seed example" is kept constant and the individuals in the block are generated following the ecuation:

$$D(E_i, E_j) \leq \delta \quad (9)$$

where δ is a value computed each generation for each SI, depending on the fitness value for that individual. The computation of δ values follows:

$$\delta = \frac{b^2(e^{[\frac{2\ f\ ln(\frac{a-b}{b})}{F_{Max}-F_{Min}}]} - 1)}{a - 2b} \quad (10)$$

Where a is a scale constant to constrain the maximum value of the increment, b is another constant to regulate the shape of the function, F_{Max}, F_{Min} are the best and worse fitness respectively, and f is the fitness of the individual.

The δ values are used to generate examples which distance from one each other is δ.

In figure 4, an example of the shape of the increment function is shown, for $F_{Min} = 0$.

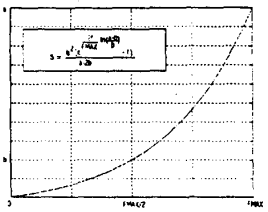

Figure 4: Function to compute the Incremental Genetic Operator.

Evolution of population.- Initially, all the seed examples of the blocks are identical and randomly generated. The individuals in the blocks are all the same and,in this initial step, equal to their related seed example.

For the evolution of examples, the following rules are used:

- Generate ramdomly the first example (seed example) for the individual I (SI_i)
- Generate the rest of individuals by the equation:

$$N(0, I_j) = D(E_i, E_{i+1}) \quad (11)$$

  Where $N(0, I)$ is a Normal distribution, means 0, and deviation I, and $D(A, B)$ is the distance between examples A and B.
- Crossover the blocks of examples. When a new SI individual is generated by crossover, the examples of this new SI are inherited from its parents by crossing-over the blocks of examples of the parents.

The dynamic of the whole system is as follow:

- Initially a set of M random $SI's$ are generated.
- For each one of the above $SI's$ a set of N seed examples are also randomly generated. One for each block.
- The fitness of the SI is computed over each individual in its related PE population.
- The final fitness value for the SI is computed using the equation 1.
- The fitness value for each EI individual in each population is set inversely to the fitness value of the corresponding SI when dealing with the EI.
- For the $SI's$ an evolutionary method is used to compute the followings generations.
- The offsprings inherit the population of $EI's$ from theirs parents, mutated in the way above described.

4. The majority function in one-dimensinal CA

To check the capabilities of the model, a classical problem in the field of learning rules for Cellular Automata (CA) has been selected: the majority function in one dimensional CA [7].

A one dimensional CA is a linear array composed of N cells that can take one of k possible states. A rule is defined to update the state of the cells. The rule decides the state of a cell based on its previous state and the state of their surrounding cells. The problem is to find a rule able to set all the cells of a binary CA to 0's or 1's depending on the density of the initial configuration. If the CA has more 0's than 1's in the initial configuration, the rule has to set all the cells to 0 in a finite period of time, ant to 1 elsewhere. No rule is known able to solve the problem for all possible initial configurations. There are some rules able to solve the problem for an important number of initial configurations. If the density of 0's in the initial configuration (ρ_0) is high or short, the problems become an easy problem. Since ρ_0 goes near $\frac{1}{2}$ the problems arise in difficulty [8].

There are some works that use Genetics Algorihtms [9, 10], or Genetic Programming [11], to explore the space of rules, for this kind of CA. Even some schematas of coevolution have been used for this problem [12] showing the difficulty of coevolving two populations towards continuous improvement.

In this work we are showing the results achieved in the majority function problem using a simple genetic algorithm and the uniform coevolution model. First a simple GA without coevolution has been used for comparision purposes. Two different schematas of GA have been implemented, with two different computations of the fitness function:

- **Simple fixed GA**.- In this case the computation of the fitness function is the result of the evaluation of a rule over some fixed initial configurations randomly generated. The initial configurations are the same for all the rule, in all generations.
- **Simple random GA**.- The computation of the fitness function is the result of the evaluation of the rule over some variable random initial configurtations. The initial configurations are changed each time a rule has to be evaluated.

These schematas have been compared with the Uniform Coevolution model. Each experiment consists on 25 evolutions of the model, with a different random seed. The best individual of each evolution has been evaluated with a test set of initial configurations. In table 4. can be seen the result of each experiment in both, learning set and test set:

	Learning	Test
Simple Fixed GA	100.0	80.84
Simple Random GA	100.0	91.96
Uniform Coevolutive GA	100.0	92.26

Table 1:

All the experiment show a high adaptation level in training period. The main difference, if observed in more detail, is that the experiment with fixed training examples shows a faster adaptation. The advantage of Uniform Coevolution arises when generalization capabilities are observed. The Uniform Coevolution is able to solve 0.3% more test cases, in average, that random and 11.42% more than fixed experiments.

In order to test the roboustness of the method to find good solutions, the figure 5 has been included:

In this figure the probability of finding a solutions is shown. In the x-axis the goodness of the solution is plotted,

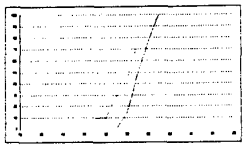

Figure 5: Probablilitie of finding solutions.

and in the y-axis the probability of the method to find a solution of, at least, that goodness. Figure 5 corresponds with test set. It is clear that in the random method the shape of the plot is more abrupt. This means that the probability of finding a solution able to solve more than 90% of the test cases is much higher. This high level of generalization is due to that the random generation of training cases slows down the learning process in training period, and makes more general the achieved solution. This is clearer with Uniform Coevolution. The probability of finding better solutions is even higher. The specific adaptation to each solution of the training cases makes the method to converge faster and more accurate to the generalization capabilities.

Finally, the evolution of the generalization capabilities of the best individual of each method is shown in figure 6.

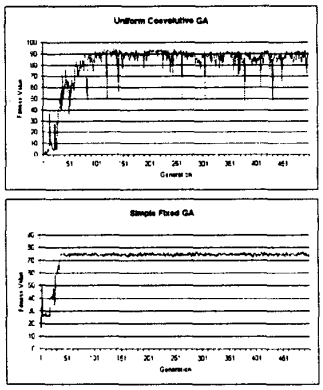

Figure 6: Evolution of generalization capability.

The evolution of the generalization capabilities for fixed training examples method is gradual, slow and reachs a local minimum very at the beginning of the evolution. By the opposite, the the generalization capability for uniform Coevolution Method grows faster and reach a higher value. The irregularities in the plot coresponds to the involution process in which some individuals takes advantage of the evolution of easier learning sets for them. This individual disapears in next generations as a result of the coevolution process. However, this mechanism allows to explore new, and some times better, search space areas.

5. Conclusions

It is clear that coevolution is a good paradigm to solve the testing problem. The adaptation of particular training examples for each specific solution can help in the faster and more accurate learning of solutions. Usually, to fix the learning examples makes the method to over-adapt to that examples. This is more evident in evolutionary computation methods. By the opposite, to generate newer examples set, each evaluation period makes harder the evolution of solutions. In some cases the method becomes inapplicable.

Uniform Coevolution overcomes these difficulties by evolving independent examples data sets for each particular solution. Moreover, the data set becomes gradually more complex as the solution becomes more accurate. The experiments show the learning and generalization capabilities of the method, that is able to solve better and faster a hard CA problem.

The paper present the Uniform Coevolution as a new framework of coevolutionary learning, that could be implemented in many existing learning methods. Here, a GA has been used as the main learning method for Uniform Coevolution, but it could be included in other techniques, i.e. Genetic Programming, Evolutionary Strategies, or some simbolic learning methods.

6. References

[1] P. R. Ehrlich and P. H. Raven. Butterflies and plants: a study in coevolution. *Evolution*, I(18):586–416, 1964.

[2] J. Roughgarden. Resource partitioning among competing species: a coevolutionary approach. *Evolution*, I(18):586–416, 1964.

[3] R. Dawkins and J.R. Krebs. Arms races between and within species. In *Proceedings of the Royal Society of London B*, pages 489–511, 1979.

[4] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C.G. Langton, editor, *Artificial Life II*, pages 313–324, Reading, MA, 1991. Santa Fe Institute, Addison-Wesley.

[5] J. Paredis. Coevolutionary computation. *Artificial Life*, 2:355–375, 1996.

[6] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary computation*, 5(1):1–29, 1997.

[7] S. Wolfram. *Cellular Automata and Complexity.* Addison-Wesley, 1996.

[8] M. Mitchell, P.T. Hraber, and J.P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[9] M. Mitchell, J.P. Crutchfield, and P.T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[10] R. Das, M. Mitchell, and J.P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In *Parallel Problem Solving from Nature III*, volume 866, pages 344–353. Springer-Verlag, 1994.

[11] D. Andre, F.H. Bennett III, and J.R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Fifth Artificial Life Conference*, pages 16–18, 1996.

[12] J. Paredis. Coevolving cellular automata: Be aware of the red queen! In *Seventh International Conference on Genetic Algorithms*, page 393.400. Morgan Kaufmann, 1997.