Coevolutive adaptation of fitness landscape for solving the testing problem

A. Berlanga, P.Isasi, A. Sanchis and J. M. Molina

Departamento de Informática, Universidad Carlos III de Madrid
Avda Universidad 30, 28911- Leganés (Madrid)

ABSTRACT. In this work a general framework, called Uniform Coevolution, is introduced to overcome the testing problem in evolutionary computation methods. This framework is based on competitive evolution ideas where the solution and example sets are evolving by means of a competition to generate difficult test beds for the solutions in a gradual way. The method has been tested with two different problems the robot navigation problem and the density parity problem in cellular automata. In both test cases using evolutive methods, the examples used in the learning process biased the solutions founded. The main characteristics of the Uniform Coevolution method are that smoothes the fitness landscape and, that gets the "ideal learner examples". The results using uniform coevolution show a high value of generality, compared with no Co-evolutive approaches.

1. INTRODUCTION

The coevolutive dynamics evolve a solutions system and an examples system. In "learning by examples methods", the performance of the solutions depends of the examples used. In coevolutive methods, the evolution of the examples tries always to generate harder examples for the solutions. As the solutions are more complex and accurate, they must prove their capabilities with more sophisticated and complex examples [1], [2].

In many problems, the generation of good solutions over reduced examples set is a very difficult task. In this case:

- If the examples evolve toward hard data sets, the process could end into an impossibility of achieving solutions for these hard example sets, and the continuous adaptation of the examples could stop the adaptation of the solutions. These are cases in which the fitness landscape is abruptly sharpened by the examples.

- If the adaptation of examples process is carried out in such a way that the adaptation of solutions is allowed, the solutions could reach in a process of over-adaptation, making more difficult the generation of more accurate solutions with a high value of generality. In these cases, the examples are not modified too much in order to allow the generation of solutions, but the generated solutions become good for the particular examples set, and solutions are not able to solve the problem for different examples.

Usually, the solution obtained with a coevolutive approach in problems with huge search an examples spaces, has no idea how good is the solutions founded. The validation process is an additional problem. This is referred in the literature as the testing problem. In some previous works [3],[4],[5], we have studied evolutionary systems and having founded the testing problem because of the over-adaptation of solutions. In these cases, some rules (or neural networks) have been evolved for navigation problem in robotics (or learning rules for neural networks have been found) with a low value of generality.

To overcome the testing problem Rosing and Bellew [6] suggest a new Co-evolutionary method, the shared sampling. In this method a population of examples is always kept. Each example of the population is evaluated computing its performance over a previously selected set of solutions. In the same way, each solution is evaluated computing its performance over a previously selected set of examples. The selection of the examples is carried out proportionally to the evaluation of examples. The examples with better general evaluations are preferred as test cases for the solutions, and their evaluations are computed again. This method has the problem of a high computational cost, m solutions and n examples require $2 \times m \times n$ evaluations.

Hillis's solution [7] 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 is in continuous evolution. Is hard to compare the success of the solutions because the fitness value is too relative, a solution is tested with an example.

We propose, in this work, a new method of adjusting coevolution to allow both the evolutions of good solutions and hard test examples in difficult generalization problems. This method has been tested in two different problems where the generality of solutions is very necessary.

2. COEVOLUTIVE ADAPTATION OF FITNESS LANDSCAPE

The architecture of the Uniform Coevolution 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), see Figure 1. This structure reflects what it was called Independent Examples Sets.

The solutions and examples systems are named respectively:

- Solutions Generator System (SGS). A population of solution individuals (SI) composes it. For computing each SI fitness, is necessary to face each individual with a set of different situations, examples, represented by a population in the Examples Generator System.

- Examples Generator System (EGS). It is a meta-population composed of meta-individuals, which are populations of examples (PE). Each PE is related with a SI. Example individuals (EI) compose the PE. The fitness of those individuals is inversely proportional to their related SI'

0-7803-6583-6/00/$10.00 © 2000 IEEE

fitness, when operating over them.

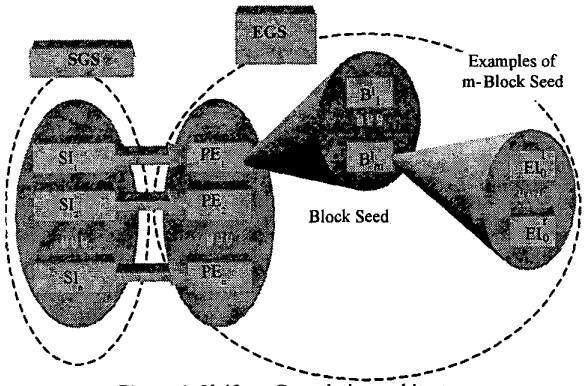


Figure 1: Uniform Coevolutive architecture.

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

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

Solution Generator System

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

The generation of better solutions is driven by the evaluation function, also called 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. As previously mentioned, for the smoothing fitness landscape mechanism, a linear combination of evaluations is used as fitness value of the individual. The fitness of an individual i is computed using the evaluation values of that individual over a set of n examples, in equation 1.

$$F(SI_i) = \sum_{j=1}^{nb} \frac{F_B^j(SI_i)}{nb} \pm C\sigma_B^i \qquad (1)$$

Where $F(SI_i)$ is the final fitness of the *i-th* solution individual, *nb* is the number of blocks in the population of examples, $\sigma^B{}_i$ is the deviation of the fitness values of the blocks for SI_i, C is a constant measuring the importance of the deviations over the normalized total fitness of the blocks, and $F_j^B(SI_i)$ is the fitness of SI_i for the block j. This value is computed following equations 2 and 3.

$$F_B^j(SI_i) = \sum_{k=1}^{nex} \gamma_{ik}^j f(SI_i, EI_{ik}^j) \qquad (2)$$

$$\gamma_{ik}^j = \frac{w_{ik}^j}{\sum_{k=1}^{nex} w_{ik}^j} \qquad (3)$$

Where $f(SI_i, EI^j_{ik})$ is the value of the evaluation of SI_i over the example EI^j_{ik}, *nex* is the number of examples of each block. The w^j_{ik} values are used to weight the importance of each example in the total computation of the fitness of a SI. The w values depend on the proximity between the fitness they are weighting and the maximum fitness, and they are computed by the equation 4.

$$w_{ik}^j = \frac{\left(e^{a_i}-1\right)\left(e^{\beta_j^i}-1\right)+\left(e^{1-a_i}-1\right)\left(e^{1-\beta_j^i}-1\right)}{(e-1)^2} \qquad (4)$$

Where a_i is a measure of the evolution degree of the individual over its examples set, and the β_j^i gives an idea about how the example j contributes to the total fitness of individual i. The a_i values are computed by equation 5.

$$\begin{cases} a_i = 1 + \dfrac{\overline{fit^i} - F_{MIN}}{F_{MAX} - F_{MIN}}, & \text{if minimize} \\ a_i = \dfrac{\overline{fit^i} - F_{MIN}}{F_{MAX} - F_{MIN}}, & \text{if maximize} \end{cases} \qquad (5)$$

$$\text{where} \quad \overline{fit^j} = \frac{\sum_{i=1}^{nex} f(SI_i, EI_{ik}^j)}{nex}$$

And β_j^i by equation 6.

$$\begin{cases} f_{i,max} = f_{i,min}, \ \beta_i^j = 0 \\ f_{i,max} \neq f_{i,min}, \ \beta_i^j = 1 - \dfrac{f(SI_i, EI_{ik}^j) - f_{i,max}}{f_{i,min} - f_{i,max}}, & \text{if minimize} \\ f_{i,max} \neq f_{i,min}, \ \beta_i^j = \dfrac{f(SI_i, EI_{ik}^j) - f_{i,max}}{f_{i,min} - f_{i,max}}, & \text{if maximize} \end{cases} \qquad (6)$$

Where F_{max} is the maximum fitness value that a *SIi* could ever reach, $f_{i,max}$, $f_{i,min}$ are the maximum and minimum fitness values achieved for SI_i over its related examples set respectively, described through equations 7 and 8.

$$f_{i,max} = Max\{f(SI_i, EI_{ij})\}_{j=1}^{N} \qquad (7)$$

$$f_{i,min} = Min\{f(SI_i, EI_{ij})\}_{j=1}^{N} \qquad (8)$$

Examples Generator System

The second subsystem in Uniform Coevolution is called Examples Generator System (EGS). The EGS is a meta-population composed of a set of populations $\{PE_i\}$. Therefore, the EGS is composed of two dependent evolutive systems: the meta-population $\{PE_i\}$ and the PE_i, one embedded into the other.

(a) Meta-population. As a way of developing the independent examples sets idea, the examples are divided in M independent sets PE_i, which are the individuals in the meta-population. Each

PE_i is related and competes against a unique solution SI_i. Individuals PE_i is 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 their genetic material, the blocks are interchanged.

(b) Population PE_i. All the 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 unique method for evolving PE_i.

(c) 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 equal, in this initial step, to their related seed example. In furthers steps of evolution, the individuals in a block are generated from the seed example by the Incremental Genetic Operator (IGO). The blocks of PE_i are inherited by the offspring from their parents.

(d) Incremental Genetic Operator. For the designing of the IGO is necessary to define a distance function between examples. This distance is a measure of the differences existing among examples: most different are two examples a higher value outputs the function and vice-versa (equation 9).

$$D: E \times E \rightarrow \Re \tag{9}$$

Where E is the set of all possible examples for a particular problem.

As the distance between examples is a numerical value, the change in the examples could be computed using the equation 10.

$$I = A\left(1 - \frac{1 - e^{-\frac{2}{F_{Bmax}} \ln\left(\frac{A-B}{B}\right)\left(F_B - F_{Bmin}\right)}}{1 - e^{-2\ln\left(\frac{A-B}{B}\right)}}\right), \text{if minimize}$$

$$I = A\left(\frac{1 - e^{-\frac{2}{F_{Bmax}} \ln\left(\frac{A-B}{B}\right)\left(F_B - F_{Bmin}\right)}}{1 - e^{2\ln\left(\frac{A-B}{B}\right)}}\right), \text{if mazimize} \tag{10}$$

Where A and B are two constants to regulate the shape of the function. This shape conforms how different the examples have to be inside a block from the fitness of the individual. I values are used to generate examples which distance from one each other is precisely I.

For the evolution of examples the following rules are used:

- To generate the first example for the individual I (SI_i)
- To generate all the individuals in the block, the equation 11 is used.

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

Where $N(0,I)$ is a Normal distribution, means 0, and deviation I, and $D(x,y)$ is the distance between examples x and y. In other words, the examples are generated in such a way that their distances follow a Normal distribution of deviation I_j computed for SI_j.

3. EXPERIMENTS IN ROBOT NAVIGATION

The robot navigation problem consists on reaching a goal in a complex environment while avoiding obstacles found in its path.

A. Evolving Controllers by means of Evolutionary Strategies

It has been proven that by means of connections between sensors and actuators, a controller is able to solve any autonomous navigation robotic behavior [8]. This theoretical approach is based on the possibility of finding the right connections of a feed-forward Neural Network, NN, without hidden layers for each particular problem, see Figure 2. The input sensors considered in this approach are the ambient and proximity sensors, s_i, of Figure 3. The NN outputs are the wheel velocities.

The NN architecture is shown in Figure 2.

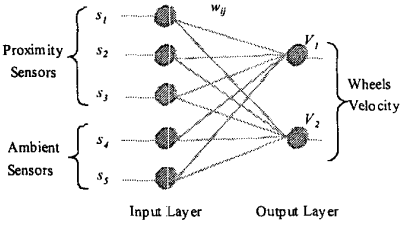
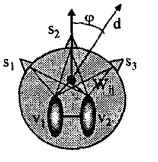


Figure 2: Neural Network Controller.



s_i: Input of i-sensor
v_j: Velocity of j-wheel
d: Goal distance
φ: Goal angle
W_{ij}: Weight between i-sensor and j-wheel

Figure 3: Connections between sensors and actuators in the Braitenberg representation of a Khepera robot.

The velocity of each wheel is calculated by means of a linear combination of the sensor values, using those weights (Figures 2 and 3) as shows equation 12.

$$v_j = f\left(\sum_{i=1}^{5} w_{ij} \times s_i\right) \tag{12}$$

Where w_{ij} are searched weights, s_i are sensor input values and f is a function for constraining the maximum velocity values of the wheels.

Weight values depend on problem features. To find them automatically, an ES is proposed [9]. In this approach each individual is composed by a 20 dimensional-real valued vector, representing each one of the above mentioned weighs and their corresponding variances. The individual represents one robot

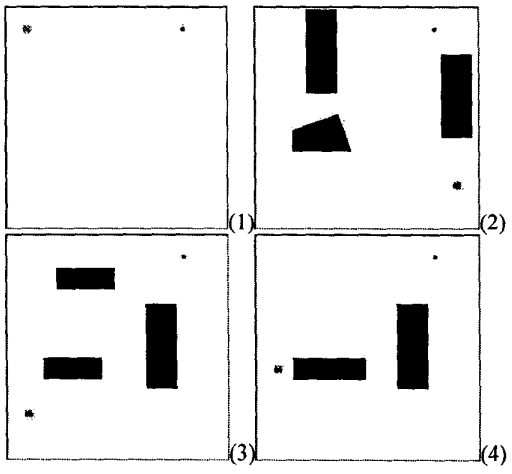
behavior consequence of applying the weights to the equation 12. The evaluation of behaviors is used as fitness function.

B. Environment

In this work, a simulator based on an autonomous robot named Khepera [10] is used. The mini-robot Khepera is a commercial robot developed at LAMI (EPFL, Laussanne, Switzerland). Experiments take a long time of continuous functioning of the hardware. In order to prove the different configurations of the controllers, a simulator developed in a previous work [11] has been used, the SimDAI one. In the simulator, the characteristics of the turtle robot model [12] and the physical restrictions of the Khepera robot have been considered. SimDAI is a working prototype of a mobile robot simulation environment for experimenting with robot navigation and control algorithms. Each mobile robot is completely independent, can navigate and interacts with other robots in a 2-D simulated world of obstacles, which is separately monitored. The simulation world consists of a rectangular map of user defined dimensions, where particular objects are located. In this world it is possible to define a final position for the robot.

C. Robot Navigation Results

Two different kinds of experiments have been performed. In both cases, an Evolutionary Strategy [13], [14], is used, $(\mu+\lambda)$-ES, $\mu=6$, $\lambda=4$, in order to find the network connections weights. Experiments differ in the way they are evaluated on the learning environments. One of the experiments, which will be referred as *fixed*, is trained in the same environment during all the evolutive process; that is, starting and goal positions, as well as the obstacle configuration are constant. On the other hand, those experiments that use the uniform coevolution algorithm, *coevU*, evolve the robot starting position and orientation, while they keep fixed the goal position and obstacles configuration. Figure 4 shows the training environments. The objective of the evolutive process is to minimize the fitness value.



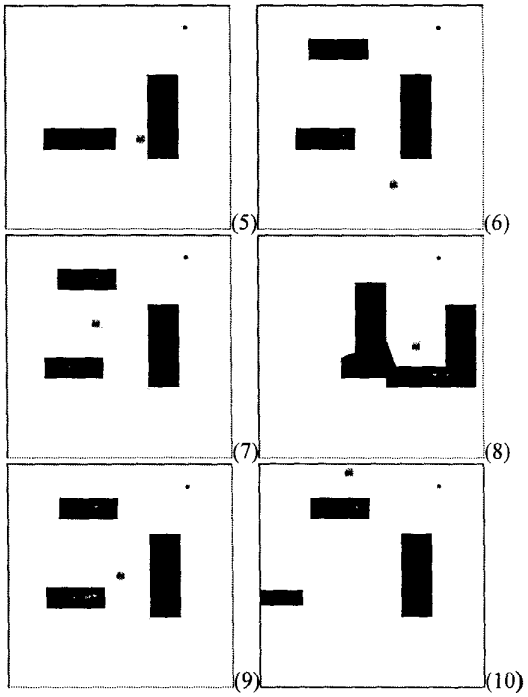


Figure 4: Evaluation environments. The little point represents the goal and the big point represents the robot starting position.

Measure of the controllers fitness

To obtain controller fitness value, the simulation has been run for a period of 2000 cycles. Simultaneously, a log of its behavior is recorded. The measures that will be taken into account to calculate the fitness value are the following:

- Number of cycles necessary to reach the goal, T. If the goal is not reached, the value is 2000.
- Length of the robot's trajectory, L.
- Number of collisions, C.
- Number of cycles in which the robot stayed in the same position, S.
- Euclidean distance between the robot's starting and final position, D_o.
- Euclidean distance between the robot's starting position and the goal position, D_m.

Equation 13 shows the lineal combination and weights used to compute the fitness value of a controller, experimentally obtained from the measurements of its behavior.

$$f_j^i = 20T - 1.5L + 10C + 10S + 10D_m - 1.5D_o \quad (13)$$

For the *fixed* experiment the fitness function is the base measurement used to apply the selection operator. For the *coevU* experiment, this is the value, $f(SI_i, EI^j_{ik})$, applied in equations 2, 6, and 8 to calculate the block fitness value. In these experiments, constant C in equation 1 has an experimental value of 0.25.

Results

The fixed type experiments have two main problems: the overadaptation problem and the quality of the solutions that depends on the training examples set. Thus, the necessity of an evolutive algorithm to improve the existing one is justified.

Coevolutive experiments have not been performed in all the environments since some of these only differ in the starting position, thus for example 3, 6, 7 and 9 in figure 4 are the same environment.

The fitness values of $fixed_1$, $fixed_3$ and $CoevU_1$, $CoevU_3$ are related with the evolution in worlds 1 and 3. These two worlds are the most general ones from figure 4. In table 1, the validation of the obtained controllers is shown. These controllers have been learned in worlds 1 and 3 (of figure 4) and tested in worlds 1, 3, 5 and 10.

Table 1: Resume of the results in *fixed* experiments and *CoevU* experiments.

		W_1	W_3	W_5	W_{10}
$Fixed_1$	Fitness	66.4	89.7	87.0	79.4
	σ Fitness	15.5	19.8	21.1	21.1
$Fixed_3$	Fitness	73.2	95.5	95.0	91.2
	σ Fitness	15.9	9.7	9.6	14.0
$CoevU_1$	Fitness	1.9	60.6	67.2	31.7
	σ Fitness	1.0	35.7	31.0	40.8
$CoevU_3$	Fitness	5.6	21.7	20.6	6.3
	σ Fitness	4.6	15.3	24.2	9.9

The validation process has been carried out making 1000 executions over worlds 1, 3, 5 and 10. Each execution has different initial position and orientation of the robot, randomly generated.

Table 1 compiles the average over the 1000 running. The controller denoted through $CoevU_1$ shows the best generalization results, better than any one of the fixed controllers. Moreover, it also can be seen that $CoevU_1$ has a very specialized behavior in the world 1, comparing with the results of $CoevU_3$. This last controller shows better results in the four worlds considered. $CoevU_3$ improves the fixed controllers in about an 80% and about a 20% over $CoevU_1$.

4. EXPERIMENTS FOR THE DENSITY CLASSIFICATION PROBLEM

Cellular Automata (CA), are spatially-extended discrete dynamical systems whose architecture has many desirable features. CA performs computation, with local interactions, in a distributed fashion on a spatially-extended lattice, [15].

The Density Classification Problem (DCP) is one of the most studied problems in Cellular Automata [16]. This problem is interesting from both, theoretical and practical aspects, and it has been proven the non-existence of any rule able to solve the problem for a binary CA with a neighborhood of radius one [17]. The DCP is defined by the equation 14.

$$T_{\rho_c}(N,M)=\begin{cases} <\Psi_m(s_0)=0^N \text{ if } \rho(s_0)<\rho_c \\ \Psi_m(s_0)=1^N \text{ if } \rho(s_0)>\rho_c \\ \text{not determined if } \rho(s_0)=\rho_c \end{cases} \quad (14)$$

Where $T_{\rho c}(N,M)$ is an unidimensional DCP of size N, with a critical density of ρ_c and after M updating periods. If the initial density $\rho(s_0)$, is shorter than the critical density, the CA has to transit, after M steps, to a configuration of all zeros. s_0 is the initial configuration, the configuration of a CA after some i steps is $s_i=\Psi(s_0)$, where the function Ψ defines the rule of the CA.

A genetic algorithm (GA) has been used to evolve the SGS. In the EGS the initial state configuration, needed to measure the performance of CA rules, are codified. The objective is to obtain a CA rule with the highest average of right classifications.

Six kinds of experiment have been performed. In table 4 is shown a brief description of each type of experiment. The examples set column identifies the training examples generation. "Fixed" means that always the same examples are faced for all the individuals. "Random" means that a new random examples set is generated each generation for each individual in the population. "Coev" means that the example sets are generated following the previously mentioned rules for the Uniform coevolution method. The word "UNI" means that the Uniform Coevolution method is applied completely. When "NoSPC" appears, no Selective Pressure Control is used. The word "NI" is related with the local fitness function used. Two fitness functions have been introduced named informed fitness function and non-informed fitness function:

- Non-Informed fitness function. It is computed by the percentage of success of the CA rule over the examples set.
- Informed fitness function. In this case some domain knowledge is taken into account. It is known that initial configurations with a density near 0.5 are more difficult to classify and better to achieve generalized rules. This fact is used to overweight these especially difficult initial configurations, to introduce a selective pressure toward more general CA rules.

A total of 30 experiments of each type have been realized to overcome the stochasticity of the GA. In all the experiments a GA have been used to evolve the solutions. In tables 2 and 3 the GA and CA parameters of runs are shown.

Table 2: GA parameters.

Population size	100
Chromosome length	128
Crossover probability	0.9
Mutation probability	0.01
Elitism	0.05
Generations	500
Selection operator	Tournament (3)

Table 3: CA parameters.

States	Binary
Neighborhood	7
Dimension	Uni-dimensional
Number of rules	3.4×10^{38}
Boundary conditions	Periodic
Lattice size	149
Number of initial configurations	7.1×10^{44}

These initial configurations are the same for all the experiments and are equally distributed in ten density intervals. This supposes one hundred configurations of density between 0.0 and 0.1, one hundred between 0.1 and 0.2 and so on. The percentage of successfully classified configuration is the measure of generalization of the rule.

Table 4: Generalization results

Experiment	Best Value	Average Value	Sigma
FixedNI	93.46%	91.68%	0.92
Fixed	93.98%	70.74%	20.43

RamdomNI	93.01%	92.10%	0.27
Random	94.10%	92.55%	0.51
CoevNoSPC	93.48%	92.55%	0.42
CoevUni	93.61%	92.63%	0.51

To test the generalization capabilities of the obtained CA rules a total of 1000 new initial configurations are generated. The results of this generalization test are summarized in table 4. There are not fundamental differences in best values for all the experiments, however from the average values it can be inferred some conclusions. For the *FixedNI* experiment the average value is shorter. In this case some times the solutions correspond with a local minimal in generalization. This is due to a problem of over-fitting. With the elimination of information in the fitness function, the learning process become less accurate, and in generalization the over-fitting problem can be overcome. Besides, the Co-evolutive experiments show a better generalization capability, and the inclusion of the SPC mechanisms helps to achieve this higher generalization level. These best results are achieved even without including any domain dependent information in the fitness function for the Co-evolutive experiments. It is expected that with such information the results could be improved.

In stochastic methods, the information of the best solution found can be non-realistic. To really compare the efficiency of some methods is better to have an estimation of the easiness of finding a solution in one run. For the Co-evolutive experiments all the solutions have a value of generality higher than 92.5%, and there is a probability of 0.1 of finding solutions of above 94%. It could be notice that the evolution of individuals is faster for non informed fitness function. This implies that is easy to adapt to the training set when information is given to the fitness function. However, some times the results are stacked in sub-optima and the peaks in the figure reflect these situations (remember that the results show the average of the fitness values over the whole set of experiments of each type). In the case of Uniform coevolution two main goals have been achieved. On one hand, the learning process has been slowed to avoid over-adaptation and to allow generalization. On the other hand, there is less variations in the solutions, so there is a high uniformity in the solutions achieved. The evolution of the examples reduces the diversity of suboptimal solutions, allowing generality (contrarily of what happen in random examples sets).

5. CONCLUSIONS

This work has shown some of the important disadvantages of evolutionary computation techniques, especially in complex problems where general solutions are needed in learning through examples. The new method proposed in this work present these characteristics: (a) Avoid overadaptation. The coevolutive mechanisms proposed slowly down the adaptation of solutions during learning. (b) Keeping on the genetic diversity. Usually the evolutionary computation methods converge prematurely by faster loosing genetic material, while UC is able to keep the genetic diversity (c) Examples evolution. The simultaneous evolution of both solutions and examples allow the automatic and gradual generation of good training examples.

Uniform Coevolution method has been compared with a non-coevolutive algorithm to solve both the robot navigation problem and the density classification problem in CA. Their characteristics, evolution of the examples and the solutions sets and control of evolutive process, have proven the capability of the proposed system to obtain better-generalized solutions in examples-based problems.

6. REFERENCES

[1] P.J. Angeline and J.B. Pollack, Competitive environments evolve better solutions for complex tasks Proceedings of the Fifth International Conference on Genetics Algorithms, ed. S. Forrest, Morgan Kaufmann, San Mateo, CA, pp: 264-270, 1993.

[2] C. W. Reynolds, "Competition, coevolution and the game of Tag" Artificial Life IV, ed. C.G. Langton, Addison-Wesley, Santa Fe Institute, Reading, MA, 1996.

[3] J.M. Molina, C.Sevilla, P. Isasi and A. Sanchis, "A reactive approach to classifier systems", Proceedings of the IEEE International Conference on System, Man and Cybernetics, San Diego, CA USA, 1998.

[4] R. E. Smith and B. Gray, "Co-adaptive genetics algorithms: An example in Othello strategy", Proceedings of the Florida Artificial Intelligence Research Symposium, 1994.

[5] A. Sanchis, J.M. Molina, P. Isasi and J. Segovia, "RTCS: A reactive with tags classifier system", Journal of Intelligent and Robotic Systems 27: 379-405, 2000.

[6] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution", Evolutionary computation, vol 5, pp: 1-29, 1997.

[7] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure", Artificial Life II, ed. C.G. Langton, Addison-Wesley, Santa Fe Institute, Reading, MA, pp 313-324, 1991.

[8] V. Braitenberg, "Vehicles: Experiments on synthetic psychology", MIT Press, Massachusets, USA, 1984.

[9] Berlanga, A., Sanchis, A., Isasi, P., Molina, J.M. "Neural Networks Robot Controller Trained with Evolution Strategies", Proc. of 1999 Congress on Evolutionary Computation, CEC99, 1999.

[10] F. Mondada and P. I. Franzi, "Mobile Robot Miniaturization: A tool for investigation in control algorithms", Proceedings of the second International Conference on Fuzzy Systems, San Francisco, USA, 1993.

[11] L. Sommaruga and I. Merino and V. Matellán and J.M. Molina, A Distributed Simulator for Intelligent Autonomous Robots, Fourth International Symposium on Intelligent Robotic Systems-SIRS96, Lisboa, Portugal, 1996.

[12] McKerrow P.J. "Introduction to robotics", Addison-Wesley Publishing Company Inc, 1991.

[13] I. Rechenberg, "Evolution Strategy: Nature's way of optimization", Optimization: Methodos and Applications, Posibilities and Limitations, Springer eds. Bergmann , Lecture notes in Engineering pp: 106-126, 1989.

[14] H.P. Schwefel, "Numerical Optimization of Computer Models", John Wiley & Sons, New York, 1981.

[15] M. Mitchell and J.P. Crutchfield and P.T. Hraber, "Evolving cellular automata to perform computations: Mechanisms and impediments", Physica D, vol 75, pp: 361-391, 1994.

[16] J. Paredis, "Coevolving cellular automata: Be aware of the red queen!", Proceedings of the Seventh International Conference on Genetic Algorithms, pp: 393-400, Morgan Kaufmann, 1997.

[17] M. Land and R.K. Bellew, No perfect two state cellular automata for density classification exists, Physical Review Letters, vol. 74, num. 25, pp:5148-5150, 1995.