Generating Training Environment to obtain a Generalized Robot Behavior by means of Classifier Systems

D. Sanchez, J. M. Molina, A. Sanchis

SCALab, Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad 30, 28911 (Leganés)-Madrid, SPAIN

Abstract

Classifier System are special production systems where conditions and actions are codified in order to learn new rules by means of Genetic Algorithms (GA). These systems combine the execution capabilities of symbolic systems and the learning capabilities of Genetic Algorithms. A Classifier System is able to learn symbolic rules that allow to react to new environmental situations. The capacity of CS to learn good rules have been proven in robotics navigation problem. In this work an anylisis of the generalization capabilities of CS as functions of the testing enviornment has been devoted. Results show the improvement of the CS behavior when the training environments change.

1. INTRODUCTION

Classifier Systems (CS) [1, 2, 3, 4, 5, 6, 7] combine the advantages of rule-based systems with the possibility of applying a domain-independent learning system, such as Genetic Algorithms. The relative value of the different rules is one of the key information items to be learnt in a CS. In order to promote this learning, CS's force the rules to coexist in what is called an information-based economy service. Rules are made to compete, where the right to respond to the activation flows from the highest bidders, which will pay the value of their bids to the rules that are responsible for their activation. A chain of intermediaries is formed along this path, ranging from manufacturers (detectors) to consumers (actions to the environment). The competitiveness of the economy assures that the good (beneficial) rules survive and the bad ones disappear. There is a high level of relation and communication between the different levels of a CS [6].

The conditions and messages of a CS form a system of rules, making them a special class of production system. One of the main problems related with production systems is the complexity of rule syntax. CS's find a way around this problem by restricting each rule to a fixed-length representation. This constraint has two benefits: first, all the rules, within a restricted alphabet, are syntactically meaningful and, second, a representation using fixed-length strings allows the application of genetic-type string operators. This opens the door to search of the space of permitted rules using Genetic Algorithms [1].

As previously mentioned, traditional Classifier Systems combine rule-based knowledge representation with genetic learning. There is an obvious difference between systems that use Genetic Algorithms for learning and Classifier Systems. In the first case, the solution to the problem is fully encoded in the binary representation used by the Genetic Algorithm, that is, the evaluation of one individual is tantamount to the evaluation of the whole solution [7]. In Classifier Systems, however, the evaluation of an output is equivalent to the evaluation of a rule that partly contributes to solving the problem. This evaluation is distributed across all those rules that contribute to the activation of the end rule, using the credit reassignment algorithm [6]. In no case, however, is it an evaluation of the system composed of all the rules. This is the approach proposed by the University of Michigan [4]. New rules or sets of rules are generated from these evaluations. So, any rules that have been activated and provide a satisfactory solution to part of the problem will be the source of new rules.

Traditional CS operation is based on three fundamental concepts:

1. The solution of the global problem is a set of rules (a subset of rules is a solution to a particular situation, and a single rule may even be a solution for a very specific situation, although this is unusual).
2. Each rule's payment is distributed among the rules that activated it in the internal cycles.
3. The Genetic Algorithm allows rules to be generated from the best rules, which leads, theoretically, to an improvement in overall system operation.

The way in which Classifier Systems operate has some drawbacks, of which the following deserve a special mention:

1. With regard to the system's ability to learn chains of rules which, moreover, do not break from one learning instant to another; the loss of a rule from the chain can lead to a loss of all the knowledge due to the interrelations between rules. The rules make sense not individually but only as groups which are unknown *a priori*.
2. With regard to the need to apply the discovery algorithm to generate increasingly better classifiers and, finally,
3. With regard to the sequencing of the cases put to the system in order to guide learning towards an improvement in overall system behaviour.

The problem addressed in this paper is in particular how to combat the problem of over-adaptation to the environments in a robot navigation problem. To analyze and solve this problem, a comparison using both a classical learning process, learning from the same starting point and a sequence of randomly generated starting points is done.

Section 2 described the CS's used in this work, in Section 3, an overview of the application of a CS to the navigation problem is given, experiments are compiled in section 4 and some conclusions are founded in section 5.

2. CLASSIFIER SYSTEM

A Classifier System is composed of three main components, which can be considered as activity levels. The first level (Performance Level) is responsible for giving responses (satisfactory or otherwise) to solve the problem proposed. At this level, there are system rules, encoded by means of restricted alphabet character strings. When this level is executed, a response is given to a particular situation. The fitness of the response to the problem that is to be solved is measured by means of the reward received by the above rule from the environment. The second level (Credit Assignment) evaluates the results obtained at the lower level, distributing the rewards received by the rules that provide the output among all those that contributed to activating each of the latter rules. As this is a reinforced learning method, this evaluation can be adjusted by applying a reward or payment by the environment, whose value will be high if the solution is satisfactory and low if it is not. Reassignment can be carried out by means of different algorithms [4, 8], of which the Bucket Brigade [3] is the most commonly used and the one employed in this paper. At this level, it is not possible to modify system behaviour by changing its rules; however, it is possible to adjust their values and establish some sort of hierarchy of good and bad rules. The mission of the third level (Discovery) is to find new means for the system to discover new solutions, for which purpose a Genetic Algorithm (GA) is used.

Rules can be activated in parallel at the CS action level, whereas they are activated in series in traditional production systems. During each recognition cycle, a traditional system activates a single rule. This rule-by-rule procedure is a bottleneck for productivity growth; moreover, many of the differences between production systems architectures are related to the selection of the best strategy for activating the rule in question. CS's overcome this bottleneck by allowing the parallel activation of rules during a particular recognition cycle, or internal cycle. So, different activities can be coordinated in parallel in a CS. When a choice has to be made between mutually exclusive environmental actions or when the size of a rule has to be pruned to adapt its length to that of the message list, these decisions are left until the last possible moment when they are selected competitively, for example. So, the sequence of operations of a traditional CS can be outlined as shown in Table 1.

Table 1: Sequence of operations at the action level of a traditional CS.

Step	Operation
1	A *k*-length encoded message from the environment comes in through the input interface.
2	Clear messages list.
3	The environmental message is placed on the messages list.
4	All the classifiers whose condition coincides with the message will be activated. One message can activate several classifiers.
5	The activated classifiers will send their messages to the list.
6	Steps 4 and 5 will be repeated for *n* internal cycles.
7	Finally, a message will be chosen to produce the output through the respective interface.

A CS with n = 1 (see Table1) means that step 6 is not applied. Then, the CS do not allow rules chaining and all the learned rules are input-output relations. In this sense, the considered CS works in a reactive way.

3. NAVIGATION BEHAVIOR

A fundamental requirement for autonomous mobile robots is navigation. This task gets the robot from place to place with safety and no

damages. Approaches based on the classical paradigms (abstraction, planning, heuristic search, etc.) were not completely suitable for unpredictable and dynamic environments. Another approaches consider reaction as the new paradigm to built intelligent systems. One classical instance of this kind of architectures is the subsumption architecture which was proposed by Brooks [9] and has been successfully implemented on several robots of MIT and other institutes. The base of the subsumption architecture is "behavior". Each behavior reacts in a situation and the global control is a composition of behaviors. Different systems, from finite state machines to fuzzy controllers, have been used for the implementation of these behaviors. The rules of these behaviors could be designed by a human expert, designed "ad-hoc" for the problem, or learned using different artificial intelligence techniques [10].

Machine learning has been applied to shape the behavior of autonomous agents in this kind of environments. Some of these techniques become inapplicable to the learning reactive behavior problem because they require more information than the problem constraints allow. Thus, it would seem reasonable to use an automatic system that gradually builds up a control system of an autonomous agent by exploiting the changing interactions between the environment and the agent itself. Some approaches use Genetic Algorithms to evolve fuzzy controllers [11,10], Evolution Strategies to evolve connection weights in a Braitenberg approach [12, 13] or Neural Networks to learn behaviors [14].

In this work, we use a simulator (Figure 1) based on a mini-robot Khepera [14], which is a commercial robot developed at LAMI (EPFL, Laussanne Switzerland) (Figure 2). The robot characteristics are: 5.5 cm of diameter in circular shape, 3 cm of height and 70 gr. of weight. The sensory inputs come in from eight infra-red proximity sensors. These sensors are composed by two parts: an IR emitter and a receiver. The emitter and the receiver are independent, so that it is possible to use the receiver to measure the reflected light (with the emitter active) or to measure the environmental light (without emission). The reflected light measurement can give some information about the obstacles surrounding the robot. In fact, this measure is not only a function of the distance of an object in front of the emitter but also the environmental light and the object nature (color and texture). So the value of distance is modified by the measure of the ambient light and the object nature, the light use is constant and all the obstacles used have the same color and texture. The robot has two wheels controlled by two independent DC motors with incremental encoder that allows any type of movement. Each wheel velocity could be read by an odometer.

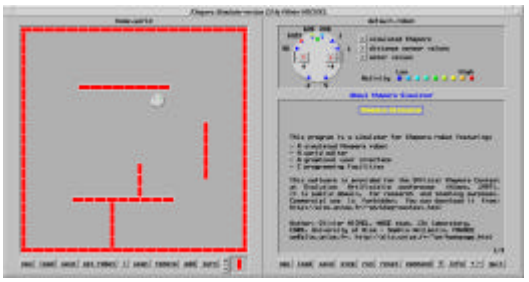

Figure 1. Khepera Simulator Environment.

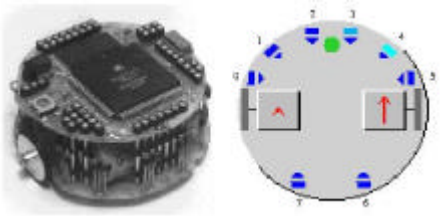

Figure 2. Khepera robot real and Simulated.

The sensors (proximity, ambient and odometer) supply three kinds of incoming information: proximity to the obstacles, ambient light and velocity. Representing the goal by a light source, the ambient information lets the robot know the angle (the angle position in the robot of the ambient sensor receiving more light) and the distance (the amount of light in the sensor) to this goal. The sensors used as input in the CS are grouped as shown in Figure 3. Figure 4 shows the complete input message of the CS. The considered values of the codification are shown in Table 2.

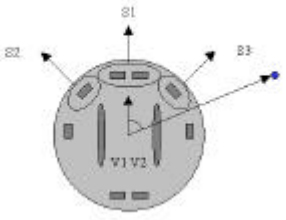

Figure 3. Grouped Sensors in Khepera Robot.

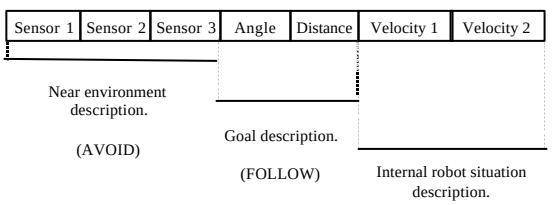

Figure 4. Grouped Sensors in Khepera Robot.

Tabla 2*:* Codification values of input message.

	00	01	11	10
Obstacle Distance (OD)	Very Near	Near	Far	Very Far
OD values	0 - 10	10 - 20	20 - 30	30 - ∞
Angle to Goal (AtG)	0 - π/4	π/4 - π	π - 7π/8	7π/8- 2π
AtG value	Sensor 3	Sensors 4, 5, 6	Sensors 7, 0, 1	Sensor 2
Distance to Goal (DG)	Very Near	Near	Far	Very Far
DG values	0 - 50	50 - 100	100 - 250	250 - ∞
Velocity (V)	Slow	Quick	Back	Stop
V values	4	8	-8	0

4. EXPERIMENTS

In this work, we present the validation of several CS learned in two different situations:

- The first type (T1) considers always the same environment in the learning phase. The number of experiments of the first type is six, all of then with the same object configuration and robot starting point (see Figure 5).

- The second type (T2) considers a random robot starting point (see Figure 5 for object configuration). Only one experiment is used but five CS (the better ones) are validated.

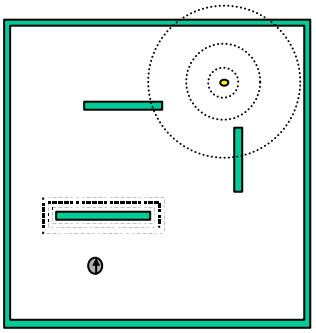

Figure 5. Learning Environment Configuration.

4.1 Learning Phase

In the learning phase the reward function is defined in equation 1.

$$P = P_{obj} + P_{obs} \qquad (1)$$

Where, P_{obj} is the reward for "FOLLOW" (go to goal) and P_{obs} is the reward for "AVOID" (surround obstacles). Each part of the reward is defined by several variables, as could be seen in equation 2.

$$P = C_{obj} * V_{obj} * P_{obj} + C_{obs} * V_{obs} * P_{obs} \qquad (2)$$

Where:

- C_{obj} is a constant to be determined experimentally.
- V_{obj} is 1, when the robot is very far from the goal; 2, when the robot is far from the goal; and 4, when the robot is near or very near from the goal.
- P_{obj} is the difference between the previous and actual distance to the goal
- C_{obs} is a constant to be determined experimentally.
- V_{obs} is 1, when the robot is far from obstacles; 4, when the robot is near from obstacles; y 8, when the robot is very near from obstacles.
- P_{obs}) is the difference between the previous and actual angle of the robot.

When the robot crashes in an object the reward is –10. Values of the two constants are obtained empirically from an ad-hoc designed CS [15, 16]. This CS is composed of three types of rules (A) (B) and (C). A type are related with avoiding very near obstacles. B type are related with follow the goal. C type are related with avoiding far obstacles. Besides random rules are added, D type.

Several environments with the same object configuration (Figure 5) have been generated to fit these constants. Starting positions of the robot are included in table 3.

Tabla 3: Starting positions of the robot.

	Coordinates	Orientation
1.	(100, 800)	0
2.	(850, 730)	180
3.	(100, 650)	0
4.	(150, 180)	270

Empirical results are summarized in Table 4. The selected values are C_{obs} = 3 and C_{obj} = 0,1 that obtain a high value for A and B type (similar in the two cases), a value near initial strength for C type and a low value for D type.

Tabla 4: Average results to fit constants.

C_{obs}	C_{obj}	A	B	C	D
3	0,1	310	315	230	120
3	0,3	280	350	220	113
5	0,1	420	210	230	123
5	0,3	380	295	225	116

Besides the fitness function, the parameters of the CS are:

- Time out to reach the goal: 2000 cycles
- Rule Number: 150
- Elitism : 0.3

- Mutation: 0.01

The genetic algorithm is applied when the robot does not reach the goal in 20 runs. A run is the execution of the robot from a starting point until the robot reaches the goal or exceeds the time out. In first type (T1), the learning phase finishes when the robot reaches the goal in all the runs. In the second type (T2), the same strategy is applied but the robot never reaches the goal in all the runs, then the learning phase finished at 2000 runs.

Validation worlds: (a) Basic World, (b) world 1 (c) world 2

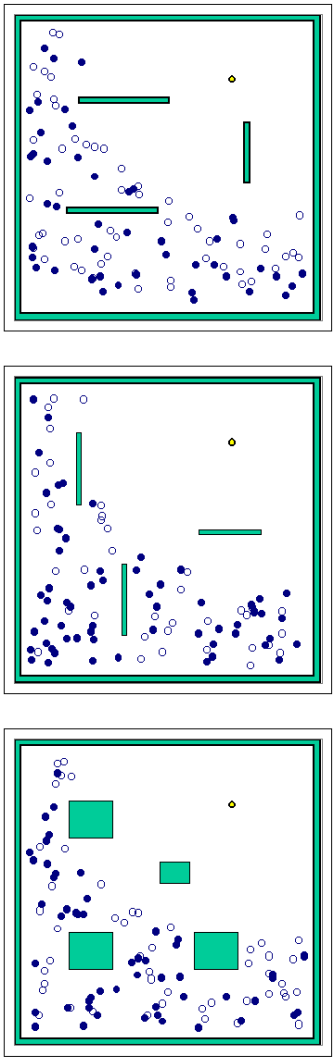

Figure 6. Validation Environments.

4.2 Validation Phase

The validation process is as follows:

- Three different object configurations: basic world, world 1 and world 2 of Figure 6.
- The CS validated are the six CS of first type (T1) and the five CS of the second type (T2). Each CS is validated using the final distribution of strength of learning phase (FS) and with initial strength assignation (IS). In this way, twelve validation of first type and ten of second one. In the case of FS no reward from the environment is needed.
- Each validation process consists in the execution of 100 runs using random starting positions far from the goal, see Figure 6.

The results of validation process are compiled in Tables 5, 6, 7 and 8.

Table 5: .Validation results T1 with IS.

INITIAL STRENGTH	Basic	1	2	Average
CS 1	55	40	48	48
CS 2	59	39	52	50
CS 3	87	64	80	77
CS 4	76	69	69	71
CS 5	68	55	52	58
CS 6	74	33	61	56
Average	**70**	**50**	**60**	**60**

Table 6: .Validation results T1 with FS.

INITIAL STRENGTH	Basic	1	2	Average
CS 1	79	53	68	67
CS 2	73	61	52	62
CS 3	85	46	72	68
CS 4	80	59	73	71
CS 5	68	51	53	57
CS 6	65	37	56	53
Average	**75**	**51**	**62**	**63**

Table 7: .Validation results T2 with IS.

INITIAL STRENGTH	Basic	1	2	Average
CS 1	88	62	73	74
CS 2	65	52	69	62
CS 3	81	72	77	77
CS 4	67	73	81	74
CS 5	91	70	76	79
Average	**78**	**66**	**75**	**73**

Table 8: .Validation results T2 with FS.

INITIAL STRENGTH	Basic	1	2	Average
CS 1	89	57	84	77
CS 2	93	70	87	83
CS 3	82	71	84	79
CS 4	82	67	69	73
CS 5	95	75	79	83
Average	**88**	**68**	**81**	**79**

The results are summarized in the following average values:

- T1 with IS: 60
- T1 with FS: 63
- T2 with IS: 73
- T2 with FS: 79

These values show that the randomly generation of starting positions improve the

generality of the final solution in the CS. This improvement could be measured as 13% in the case of IS and 16% in the case of FS.

5. CONCLUSIONS

One of the major problems related to Evolutionary Computation Techniques is overadaptation. However, the use of CS allows the system to learn gradually from training cases. Each learning process with a individual case can lead the strength to be distributed in favour of a given type of rules that would in turn be favoured by the Genetic Algorithm. If this reasoning is extended to the entire learning process, genetic diversity, which is so necessary for learning, can disappear due to the growth of a given type of rules in the population.

This is an especially serious problem when there are very different rule types in the CS. Basically, the proposed idea is to use different training cases to cover different possibilities. This strategy allows rules be more general considering different situations.

The objective of this paper was to analysis the effect of random generation of training environments in the generalization capabilities of CS.

These ideas allow the generation of general behaviors, which solve problems that require a solution with a high degree of generality. This is a typical problem when a CS controls a robot.

6. REFERENCES

[1] J. Holland, *"Adaptation in Natural and Artificial Systems"*. University of Michigan Press, Ann Arbor, (1975).

[2] J. Holland, *"Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge Bases"*, International Journal of Policy Analysis and Information Systems, vol. 4, 245-268, (1980).

[3] J. Holland, *"Properties of the Bucket Brigade"*. In Proc. of International Conference on Genetic Algorithms and their Applications, vol. 1, 1-7, (1985).

[4] J. Holland, *"A Mathematical Framework for Studying Learning in Classifier Systems"*, Physica D, 22, 307-317, (1986).

[5] J.H. Holland, *"Hidden order: how adaptation builds complexity"*. Reading Massachusetts, Addison-Wesley, (1995)

[6] D.E. Goldberg, *"Genetic Algorithms in Search, Optimization, and Machine Learning"*. Addison Wesley, Reading Massachusetts, (1989).

[7] M. Mitchell, *"An Introduction to Genetic Algorithms"*, MIT Press, Massachusetts, (1996).

[8] G.E. Liepins, M.R. Hilliard, M. Palmer and G. Ranjaran, *"Credit Assignment and Discovery in Classifier Systems"*, International Journal of Intelligent Systems, vol. 6, 55-69, (1991).

[9] Brooks R. A. "Intelligence without Representation". Artificial Intelligence, 47, 139-159, (1991).

[10] Matellán V., Molina J.M., Sanz J., Fernandez C. "Learning Fuzzy Reactive Behaviors in Autonomous Robots". Proceedings of the Fourth European Workshop on Learning Robots, Germany, (1995).

[11] Lee M.A. and Takagi H., "Integrating Design Stages of Fuzzy Systems using Genetic Algorithms". Second International Conference on Fuzzy Systems, 612-617, (1993).

[12] Isasi P., Berlanga A., Molina J. M., Sanchis A., "Robot Controller against Environment, a Competitive Evolution", Special Session on Evolution Computation, 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics. Germany, 1997.

[13] Molina J.M., Sanchis A., Berlanga A., Isasi P., "Evolving Connection Weight Between Sensors and Actuators in Robots". IEEE International Symposium on Industrial Electronics. Gimaraes, Portugal,1997.

[14] Mondada F. and Franzi P.I. "Mobile Robot Miniaturization: A Tool for Investigation in Control Algorithms". Proceedings of the Second International Conference on Fuzzy Systems. San Francisco, USA, (1993).

[15] Sanchis A., Molina J.M., Isasi P. And Segovia, J. *RTCS: a Reactive with Tags Classifier System.* Journal of Intelligent and Robotic Systems, in press. 1999

[16] Molina J.M., Sanchis A., Berlanga A. and Isasi P. *An enhanced classifier system for autonomous robot navigation in dynamic environments*. Intelligent Automation and Soft Computing. Vol. 6, 2: pp. 113-124. 2000