The Ability of Learning Automata in Solving Constraint Satisfaction Problems

P. Bahri M. R. Meybodi

Soft Computing Laboratory
Computer Engineering and Information Technology Department
Amirkabir University
Tehran Iran
meybodi@ce.aku.ac.ir

Abstract

In this paper we have investigated the performance of some stochastic methods for solving constraint satisfaction (CSP) and fuzzy constraint satisfaction problems (FCSP). The purpose of this paper is to study the abilities of learning automaton in solving these problems and comparing it with other stochastic methods. The results confirm those of [2] and show its superiority to other methods.

Keywords. CSP, FCSP, Learning Automaton, Stochastic Algorithms, Genetic Algorithms.

1-Introduction

Learning means the ability of the learner to correct its actions based on the experience of interacting with an environment. Mostly learning occurs in a stochastic environment, where there is a finite number of actions for the learner to choose. In the stochastic environment the learning agent receives responses from environment, which the agent uses to anticipate its next action. One of the simple but powerful kinds of the learning agents is the learning automaton, which was introduced by the work of Tsetlin in the early 1960's in the Soviet Union. One of the advantages of the learning automaton is that it can be implemented in a distributed manner, in order to solve the large complex systems [2].

In this paper we utilize the learning automaton to solve the constraint satisfaction problem (CSP). This problem occurs frequently in AI, and there is a vast amount of literature about the ways of solving it. In CSP there is a finite set of variables and associated with each variable is a finite set of values (labels). Moreover there is a set of constraints on choosing the values for the variables, and the problem solver searches for a consistent solution such that all constraints are satisfied.

The algorithms reported in the literature for solving constraint satisfaction problem can be classified into two groups: deterministic algorithms and stochastic algorithms. Most of the algorithms are of deterministic kind. To the author's knowledge there are very few attempts made in the literature for solving the CSP or fuzzy CSP using stochastic methods. In this paper our aim is to do a comparative study of stochastic algorithms for solving constraint satisfaction problems [2][3][4][5] with emphasis on algorithms based on learning automata. To the author's knowledge the only learning automata based algorithm reported in the literature is due to Ricci [2]. In [2] Ricci uses a collection of learning automata together with local information of the related variables to solve the constraint satisfaction problem. This algorithm will be explained later in this paper. In [3] the whole structure of the problem is used in determining the value of the variables. In [4] standard genetic algorithm is used to solve fuzzy constraint satisfaction problem as an optimization problem. In [5] algorithm reported in [4] has been improved in order to escape the local optima. In this paper we have implemented the methods reported in [2], [3], and [4] and extended the algorithm reported in [2] to fuzzy constraint satisfaction problem (FCSP). We have implemented three of stochastic algorithms with the results of simulations

coming in this paper.

It should be mentioned that the search with the learning automaton (LA) is not similar to the conventional backtracking search, in which each point in the search space is visited only once and usually some heuristics is used in choosing the values for the variables. When the search is done by learning automata, the search will not be done according to the backtracking method and also every point may be examined many times. In algorithms based on learning automata, there are a number of automata (equal to the number of variables) that act independently, i.e. they choose the values based on only their past experience and their local copy of the constraints related to the variables.

The organization of the paper is as follows. Section 2 gives an introduction to the learning automata and automata games. Section 3 describes CSP and FCSP. In section 4 how to solve CSP and FCSP by learning automata is explained. Section 5 presents the simulation results. Section 6 concludes the paper.

2- Learning Automata

Learning automata (LA) can be classified into two main families, fixed and variable structure learning automata [13]. In this section at first some of the fixed structure learning automata such as are Tsetlin, Krinsky, TsetlineG, and Krylov automata are described and then fixed structure learning automata which is used in this paper will be presented.

A fixed structure learning automaton is represented by a quintuple $\langle \underline{\alpha}, \underline{\Phi}, \underline{\beta}, F, G \rangle$ where:

- $\underline{\alpha} = \{\alpha_1, \ldots, \alpha_r\}$ is the set of actions that it must choose from.
- $\underline{\Phi} = \{\phi_1, \ldots, \phi_s\}$ is the set of internal states.
- $\underline{\beta}=\{0,1\}$ is the set of inputs where **1** represents a penalty and **0** represents a reward.
- $F: \underline{\Phi} \times \underline{\beta} \rightarrow \underline{\Phi}$ is a map called the state transition map. It defines the transition of the state of the automaton on receiving an input. F may be stochastic.
- $G: \underline{\Phi} \rightarrow \underline{\alpha}$ is the output map and determines the action taken by the automaton if it is in state ϕ_j.

The selected action serves as the input to the environment, which in turn emits a stochastic response $\beta(n)$ at the time n. $\beta(n)$ is an element of $\underline{\beta}=\{0, 1\}$ and is the feedback response of the environment to the automaton. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty probability c_i, which is the action dependent. On the basis of the response $\beta(n)$, the state of automaton is updated and a new action chosen at the time (n+1). Note that the $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action, which presents it with the minimum penalty response in an expected sense. We describe some of fixed-structure learning automata in the following paragraphs.

We summarize some of fixed-structure learning automata in the following paragraphs.

The two-state automata ($L_{2,2}$) : This automata has two states, $\phi_1$and ϕ_2 and two actions α_1 and α_2. The automaton accepts input from a set of {0, 1} and switches its states upon encountering an input 1 (unfavorable response) and remains in the same state on receiving an input 0 (favorable response). An automaton that uses this strategy is refereed as $L_{2,2}$ where the first subscript refers to the number of states and second subscript to the number of actions.

The two-action automata with memory ($L_{2N,2}$): This automaton has 2N states and two actions and attempts to incorporate the past behavior of the system in its decision rule for choosing the sequence of actions. While the automaton $L_{2,2}$ switches from one action to another on receiving a failure response from environment, $L_{2N,2}$ keeps an account of the number of success and failures received for each action. It is only when the number of failures exceeds the number of successes, or some maximum value N; the automaton switches from one action to another. The procedure described above is one convenient method of keeping track of performance of the actions α_1 and α_2. As such, N is called memory depth associated with each action, and automaton is said to have a total memory of 2N. For every favorable response, the state of automaton moves deeper into the memory of corresponding action, and for an unfavorable response, moves out of it. This automaton can be extended to multiple action automata. The state transition graph of $L_{2N,2}$ automaton is shown in figure 1.

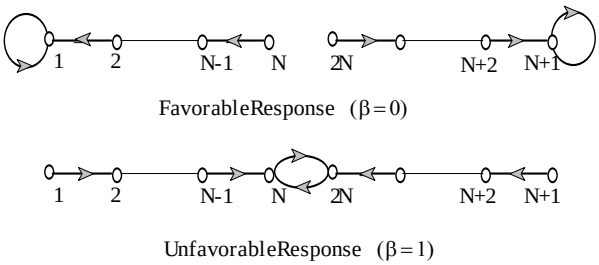

Figure 1: The state transition graph for $L_{2N,2}$

The Krinsky automata*: This* automaton behaves exactly like $L_{2N,2}$ automaton when the response of the environment is unfavorable, but for favorable response, any state ϕ_i (for i = 1, . . . N) passes to the state ϕ_1 and any state ϕ_i (for i = N+1, 2N) passes to the state ϕ_{N+1}. This implies that a string of N consecutive unfavorable responses are needed to change from one action to another. The state transition graph of Krinsky automaton is shown in figure 2.

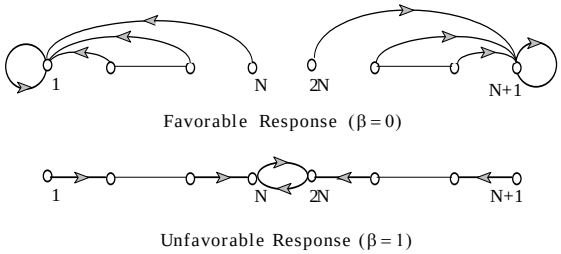

Figure 2: The state transition graph for Krinsky Automaton

The Krylov automaton: This automaton has state transition that is identical to the $L_{2N,2}$ automaton when the output of the environment is favorable. However, when the response of the environment is unfavorable, a state ϕ_i (i ≠1, N, N+1, 2N) passes to state ϕ_{i+1} with probability 0.5 and to state ϕ_{i-1} with probability 0.5. When $i \in \{1, N+1\}$, ϕ_i stays in the same state with probability 0.5 and moves to state ϕ_{i+1} with the same probability. When i = N, automaton state moves to state ϕ_{N-1} and ϕ_{2N} with the same probability 0.5. When i = 2N, automaton state moves to state ϕ_{2N-1} and ϕ_N with the same probability 0.5. The state transition graph of this automaton is shown in figure 3.

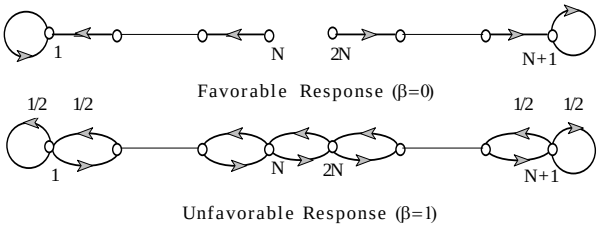

Figure 3: The state transition graph for Krylov Automaton

Variable Structure Automata: Variable-structure automaton, which will be used in this paper is represented by sextuple $\langle \underline{\alpha}, \underline{\beta}, \underline{\Phi}, \underline{P}, G, T \rangle$, where $\underline{\beta}$ a set of inputs, $\underline{\Phi}$ is a set of internal states, $\underline{\alpha}$ a set of outputs, $\underline{P}$ denotes the state probability vector governing the choice of the state at each stage k, G is the output mapping, and T is learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector. It is evident that the crucial factor affecting the performance of the variable structure learning automata is learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature [13]. Let α_i be the action chosen at time k as a sample realization from distribution p(k). The linear reward-inaction scheme the recurrence equation for updating p is defined as

$$p_j(k) = \begin{cases} p_j(k) + a\left(1 - p_j(k)\right) & \text{if } i = j \\ p_j(k)(1-a) & \text{if } i \neq j \end{cases} \quad (4)$$

if β is zero and P is unchanged if β is one. The parameter, a which is called step length, determines the amount of increases (decreases) of the action probabilities. In linear reward-penalty algorithm (L_{R-P}) scheme the recurrence equation for updating p is defined as

$$p_j(k) = \begin{cases} p_j(k) + a\left(1 - p_j(k)\right) & \text{if } i = j \\ p_j(k)(1-a) & \text{if } i \neq j \end{cases} \quad (5)$$

if $\beta(k) = 0$ and

$$p_j(k) = \begin{cases} p_j(k)\left(1 - b\right) & \text{if } i = j \\ \frac{b}{r-1} + (1-b)p_j(k) & \text{if } i \neq j \end{cases} \quad (6)$$

if $\beta(k) = 1$. The parameters a and b represent reward and penalty parameters, respectively. The parameter a (b) determines the amount of increase (decreases) of the action probabilities.

Learning automata can act jointly to do some work in a distributed fashion. A single automaton can have limited actions so the problem it tackles must be a simple one. But when we have a collection of automata each with its own set of actions, and perception of the portion of the common environment, the job complexity it can do grows exponentially in their numbers. So they are able to solve a complicated problem, whether it is a parallel, distributed, or a centralized one. The CSP and FCSP are such a problem. A system of interconnecting automata is called automata games. A game consists of more than one player interacting with an environment. In general each automata receives different responses from environment and have limited knowledge about other players status, and the decision one makes may be inconsistent with one of the others. The complexity of the game can be classified in various ways depending on the number of players (2-person, N-person) the performance criterion (zero-sum or non-zero-sum), and the nature of communications used (cooperative and non-cooperative) [1]. Our problem is N-person non-zero sum, and cooperative game. automata act in cooperation to solve a common problem. This will be illustrated in the following sections.

Learning automata have been used in many applications such as: solving NP problems [6-8], queuing theory [9], telephone traffic control [10], pattern recognition [11], control of computer networks [12-13], adaptation of parameters of neural networks [14,17], data compression [18], to mention a few. For more information about learning automata refer to [10,13]

3 -The CSP and fuzzy CSP

A constraint satisfaction problem (CSP) is a 3-tuple (X, D, C). Where $X = \{x_1, \ldots, x_n\}$ is a set of variables each x_i taking a value in the set $D_i = \{d_{i1}, \ldots, d_{ik_i}\}$, and $D = \{D_1, D_2, \ldots, D_n\}$. C is the set of constraints $C = \{c_1, \ldots, c_m\}$, such that each $c_i \in C$ contains a subset $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ of X and is defined by subset R_{c_i} of $d(x_{i_1}) \times \ldots \times d(x_{i_k})$ which determines what values of the variables are compatible with each other. A CSP is in one-to-one correspondence with a hyper graph having the same number of nodes as variables in CSP, and in which the hyper edges represents the constraints. A binary CSP is one in which the constraints are binary, so the corresponding graph has no hyper edges.

In fuzzy CSP (FCSP) each c_i represents a fuzzy relation on the Cartesian product space $D_{i_1} \times \ldots \times D_{i_k}$ characterized by a membership function $\mu_{C_i}(x_{i_1}, \ldots x_{i_k}) \in [0,1]$. A solution of a FCSP is a complete instantiation of the variables $x = \{x_1, \ldots, x_n\}$ such that all constraints C are more or less satisfied. The fuzzy set of solutions is usually denoted by as an aggregation of the fuzzy

relations defined by each fuzzy constraint i.e. $\mu_C(x) = \min_i(\mu_{C_i}(x))$ where $\mu_{C_i}(x)$ is a cylindrical extension of the fuzzy relation representing the constraint c_i to x. The objective of solving FCSPs is to find a solution (an instantiation of x) such that the global satisfaction degree of the constraints with the instantiation is maximal i.e. $\sup_x(\mu_C(x)) = \sup_x\{\min_i(\mu_{C_i}(x)\}$.

In this paper for simplicity we have assumed that the domain of each variable has equal number of values, and the graphs are binary. So all c's are such that $c \subset D_i \times D_j$ with $i \neq j$, and $i, j \in \{1,\ldots,n\}$. We say that the variables x_i and x_j are linked if there exits at least a constraint For the sake of convenience we let C be the Boolean function $C: D_i \times D_j \to \{0,1\}$ called the characteristic function of the constraint with the following condition

$$C(d_i,d_j) = \begin{cases} 1 & \text{if } (d_i,d_j) \in c \\ 0 & \text{otherwise} \end{cases}$$

As an example of the constraint satisfaction problem is n-queens problem. The problem can be described as follows: there is an $n \times n$ chessboard and we are to place n queens in the board such that no one attacks any other i.e. no two or more queens in the board are placed in the same column, or in the same row, or in the same diagonal. Another example is the graph-coloring problem. The nodes in the graph are to be colored by a fixed number of colors such that, no two adjacent nodes (i.e. nodes connected by an arc) have the same color.

In this paper instead of choosing a specific CSP problem, we have chosen random CSP's with the number of variables, and the number of values chosen from the set S={(4,4), (10,10), (12,16), (15,20)}, with the first element of the ordered pair being the number of variables, and the second element being the number of values for each variable.

4-Solving CSP and FCSP by LA

In this section we show how to solve the CSP and FCSP by a system of mutually interconnected learning automata. Each automaton is associated with a variable and actions of the automata are one of the values for that variable. A bit of notation follows:

Let $S = \{s_1,\ldots,s_k\}$ be a finite set, $\Delta(S)$ denotes the space of all probability measures on S, i.e. $q \in \Delta(S)$ iff $q = (q_1,\ldots,q_k)$, $q_i \geq 0$, $\sum_{i=1}^{k} q_i = 1$. Let $CN = (X,\alpha,C)$ be a CSP. Without loss of generality assume $|\alpha_i| = M$. Associate a learning automaton A_i with x_i at time $n \geq 0$ in which

$\alpha_i = \{\alpha_{i1},\ldots,\alpha_{iM}\}$ is the domain of x_i

$\alpha_i(n)$ is the output (action) of A_i at time n

$p_i(n) = (p_{i1}(n),\ldots, p_{iM}(n)) \in \Delta(\alpha_i)$ is the probabilities of choosing action i out of M actions (values of variables).

$\beta_i(n) \in \{0,1\}$ is the feedback to A_i at time n.

$T_i : \Delta(\alpha_i) \times \alpha_i \times \{0,1\} \to \Delta(\alpha_i)$ is the reinforcement scheme of A_i (for all i).

$\alpha_i(n)$ (action of A_i) is chosen out of α_i, according to the probability distribution $p_i(n)$, i.e. the probability that $\alpha_i(n) = \alpha_{ij}$ is $p_{ij}(n)$. All that the automata remember of their environment is kept in p_i so the space complexity of the A_i is equal to the number of values for each variable. $\beta_i(n)$ is the stochastic feedback from the environment to A_i? There are many ways to choose $\beta_i(n)$, we chosen (like [Ricci-1994]) simply as follows

$$\beta_i(n) = \begin{cases} 1 & \text{If all constraints relating } x_i \text{ are satisfied} \\ 0 & \text{otherwise} \end{cases}$$

or

$$\beta_i(n) = \prod_{j \in \gamma_i} C_{ij}(\alpha_i(n), \alpha_j(n))$$

where C_{ij} is the characteristic function of the binary constraint between x_i and x_j, and $\gamma_i = \{j \mid \exists c \in C, c \subset \alpha_i \times \alpha_j\}$. Thus every A_i is directly related with A_j that are in its neighborhood (related to A_i).

Finally T_i is the reinforcement scheme of A_i, which is central to the proper functioning of the system. $T_i = \{T_{i1}, \ldots, T_{iM}\}$, this function reads the current $p_i(n), \alpha_i(n), \beta_i(n)$ and produces $p_i(n+1)$. The model of LA we used in this paper to solve CSP is P_model which is introduced in section two.

The difference in our algorithm between CSP and FCSP is in the model of the learning automata we used for the reinforcement and the way we choose the feedback to the automata. We used S model LA for FCSP and $\beta_i(n)$ where chosen according to

$\beta_i(n) = 1 - \min_j\{\mu_{c_j}(x)\}$ where j is such c_j is related to the automaton i and the value of c_j in the variable i is equal to the value of automaton. If no c_j is satisfied we set $\beta_i(n)$ =1.

The proposed algorithm is given below.

Procedure Learning Algorithm
For $i \leftarrow 1$ **until N do**
For $j \leftarrow 1$ **until M do**
$p_{ij} \leftarrow p_{ij}(0)$
Endfor
Endfor
While (not EndCriteriaMet) and iterations <MaxIterations do
For $i \leftarrow 1$ **until N do**
$\alpha_i \leftarrow Random(p_i)$
Endfor
For $i \leftarrow 1$ **until N do**
$\beta_i \leftarrow Feedback(i, \alpha_1, \ldots, \alpha_n)$
Endfor
For $i \leftarrow 1$ **until N do**
For $j \leftarrow 1$ **until N do**
$p_{ij} \leftarrow T_{ij}(p_i, \alpha_i, \beta_i)$
Endfor
Endfor
Endwhile
Return EndCriteriaMet
End Learning Algorithm

The test function **EndCriteriaMet** is as follows:

Algorithm EndCriteriaMet
for k=0 To k less than Maximum Variable Count
if for Automaton(k) one of the action probabilities > 1.0 - ε
continue
else
return FALSE;
endif
endfor
return TRUE
End EndCriteriaMet

In the above algorithm **MaxIteration** is computed by the following formula:

Maxiterations = **a** * (number of variables) * (number of values for each variable).

Where **a** is a constant chosen here to be equal to 300. As it is said the constraints are random, that is, for each variable a random (uniform among variables) variable (not in the sense of random variable in the probability theory) is chosen and afterwards the necessary constraints are added to make the problem consistent. The number of consistent solutions is equal to #variables divided by 4.

5-Simulation Results

In addition to the proposed algorithm we have also implemented two genetic algorithms. The first genetic algorithm is for CSP and adapted from [3], and the second is for FCSP.

For the proposed solution, through simulation it is shown that except for the cases that the number of the variables and the number of values are low, we have a low probability of correct solution. This is due to the huge solution space, which makes the probability of establishing a consistent solution out of the random constraints very low. When the number of variables and values are large, the solution space becomes very large which causes the algorithm to reach the consistent solutions with low probability. Of course such problems are challenging to even the best algorithm The values of parameters ***a*** and ***b*** plays a major rule in convergence and the success rate of the algorithm as demonstrated by the results of simulation given below.

Tables 1 - 4 show the results obtained for the CSP with LA. The first table (Table 1) show the results obtained for the case where #variable = 4 and #values = 4. It is seen that large values of ***a*** produces better solutions, and as we increase ***a*** and decrease ***b*** we get better results. For this case the probabilities of correct solution is in fact rather high and for the most cases we have obtained 100% success rate, except for the cases where ***b*** is high and ***a*** is low.

In table 2, we see that as we increase ***a*** we get better performance, and for the lower values of ***a*** we have zero success rate. Furthermore as we decrease ***b*** we get better performance also, such that for ***b***=0.01 we have nearly equal performance for all values of ***a***.

In table 3, which resembles table 2, we again see that for larger values of ***a*** better rate of success is obtaied. In this case for higher values of ***b*** we have zero performance and only for ***b***=0.01 acceptable performance is obtained

Table 4 is very much like table 3. For large values of ***b*** we get very poor performance. When ***b***=0.01 as we increase a the performance becomes better. So we can conclude that for large search spaces we should keep ***b*** small and ***a*** large.

Table 1: #variables = 4, #values = 4

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	117 1%	68 64%	12 100%	3 100%	2 100%	1 100%	1 100%	1 100%	1 100%
0.1	66 67%	5 100%	2 100%	1 100%	2 100%	1 100%	1 100%	0.75 100%	0.6 100	0.6 100%
0.05	11 15%	3.7 100%	1.8 100%	1.4 100	1 100%	1 100%	0.8 100%	0.7 100%	0.8 100%	0.8 100%
0.01	2.3 100	3 100%	3 100%	3 100%	1.4 100%	3 100%	2 100%	3 100%	1.5 100%	2.8 100%

Table 2: #variables = 10, #values = 10

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	0%	0%	0%	0%	0%	97 2%	96 6%	95 13%	92 16%
0.1	0%	0%	0%	99 1%	99 7%	84 29%	71 54%	61 62%	53 68%	44 81%
0.05	0%	0%	103 11%	56 71%	20 97%	15 99%	6 100%	7 100%	5 100%	5 100%
0.01	34 92%	12 99%	14 98%	14 97%	14 97%	12 96%	11 98%	12 99%	13 97%	11 99%

Table 3: #variables = 12, #values = 16

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%
0.1	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%
0.05	0% 0%	0% 0%	0% 0%	0% 0%	35 1%	0% 0%	35 11%	35 21%	35 20%	29 46%
0.01	38 2%	9 95%	7 95%	9 91%	6 95%	9 91%	10 88%	8 92%	8 92%	8 95%

Table 4: #variables = 15, #values = 20

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%
0.1	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%
0.05	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	0% 0%	81 1%	81 1%	82 1%
0.01	0% 0%	50 68%	29 85%	26 89%	28 87%	27 89%	25 87%	30 85%	32 83%	28 88%

The results for the genetic algorithm have come in table 5. Comparing table 5 with tables 1 through 4 it is observed that the performance of the learning automaton much surpasses that of the genetic algorithm. For the first case both algorithms (for major parameter range) have the same success rate, but the difference is in the number of consistency checks (and accordingly the execution time) made by the algorithms. The genetic algorithm has two orders of magnitude higher consistency check rate than the algorithm based on learning automata. So in this respect the automaton has performed very well. The difference is more obvious when we examine other entries of the tables. The genetic algorithm has much poorer performance when the search space becomes larger. In the case that #variables=10 it has only 2 percent recognition rate, and for higher values of the parameters it has zero recognition rate with the number of consistency checks being two orders of magnitude higher. The poor performance of the genetic algorithm can be attributed to the fact that in these problems the search space is very large and at the same time the solution cases are very rare. Furthermore there is no indication of the correct answer in the nearby points of the consistent solutions. On the whole the problem is GA-hard.

Table 5: Genetic Algorithm

#Variables	#values	Success rate
4	4	30000 100%
10	10	5000000 2%
12	16	0%
15	20	0%

Table 6 through 10 shows the results of simulation for solving the FCSP by learning automata. For these simulation we have chosen the global degree of fuzzy satisfaction to be 0.6 and for the constraints, the membership function of the fuzzy relations has chosen to be a random number between 0.4 and 0.8. Moreover a consistent solution has been provided to make the problem consistent. It is seen that the results are much the same as the crisp case. In table 6 we observe that in most cases 100% success rate is obtained except for the cases where ***a*** is small and **b** is large. In table 7 again when ***b*** becomes large and ***a*** gets smaller we get poor performance. In tables 8 and 9 we have only acceptable performance for the case that ***b*** is equal to 0.01.

What can be said about the tables 6 – 9 is that when the search space becomes larg one must use very small value for ***b*** and large value close to one for ***a***. We also made similar experiments with the genetic algorithm for FCSP. The results obtaied were much the same as those for the crisp case.

Table 6: #variables = 4, #values = 4

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	117 1%	64 69%	10 100%	3 100%	1.8 100%	1.4 100%	1 100%	0.9 100%	0.8 100%
0.1	0%	72 56%	4.6 100%	2 100%	1.3 100%	1 100%	0.8 100%	0.7 100%	0.6 100%	0.7 100%
0.05	113 15%	3.7 100%	1.7 100%	1.5 100%	1 100%	1 100%	0.9 100%	0.8 100%	0.7 100%	0.9 100%
0.01	4 100%	3.3 100%	4.9 100%	1.6 100%	2 100%	2.2 100%	1.9 100%	2.5 100%	2.5 100%	2.6 100%

Table 7: #variables = 10, #values = 10

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	0%	0%	0%	0%	0%	0%	97 6%	92 15%	94 11%
0.1	0%	0%	0%	0%	100 7%	89 28%	69 51%	60 67%	53 73%	45 77%
0.05	0%	0%	100 14%	56 73%	20 96%	15 99%	7 98%	8 100%	6 100%	6 100%
0.01	40 87%	16 95%	14 97%	16 94%	10 99%	17 97%	12 98%	14 99%	12 99%	11 99%

Table 8: #variables =12, #values =16

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.05	0%	0%	0%	0%	34 1%	35 6%	35 11%	33 24%	32 33%	29 39%
0.01	38 2%	10 92%	7 95%	8 91%	94 89%	8 92%	7 92%	9 92%	8 92%	7 95%

Table 9: #variables = 15, #values = 20

A B	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
0.05	0%	0%	0%	0%	0%	0%	0%	0%	0%	82 1%
0.01	0%	58 67%	27 86%	30 81%	30 85%	17 92%	25 91%	24 91%	25 86%	26 86%

6- Conclusion

In this paper we investigated the performance of some stochastic methods for solving constraint satisfaction (CSP) and fuzzy constraint satisfaction problems (FCSP). The purpose of this paper is to study the abilities of learning automaton in solving these problems and comparing it with other stochastic methods. The results confirm those of reported by Francesco Ricci[2] and show its superiority to other methods. We intentionally have chosen the number of consistent solutions to be low, so that the probability of reaching a solution is low in order to test the ability od learning automata to situation when the search space is large. We also propose a new algorithm based on learning automata and compared it with two genetic algorithms. The simulations show that the learning

automata is a good way of solving such problems specialy when the search space is large.

References

[1] K. S. Narandra, and M. A. L. Thatachar, “Learning Automata An Introduction”, Prentice Hall, 1989.

[2] Francesco Ricci, “ Constraint Reasoning With Learning Automata”, Institute Per la Recerca Scientifica, march 14 1994.

[3] Maria Cristina, Riff Rojas, “ Using The Knowledge Of The Constraint Network To Design An Evolutionary Algorithm That Solves The CSP”, Evolutionary Computation 279-284, 1996.

[4] Ryszard Kowalczyk, “ On Solving Fuzzy Constraint Satisfaction Problems with Genetic Algorithms”, Evolutionary Computation 758-762, 1998.

[5] J. H. Y. Wong, H. Leung,” Solving Fuzzy Constraint Satisfaction Problems With Fuzz GENET”, IEEE International conference on tools with artificial intelligence, pp 174-189, 1998.

[6] Oommen, B. J. and Ma, D. C. Y. (1988). Deterministic Learning Automata Solutions to the Equipartitioning Problem, IEEE Trans. on Computers, No. 37, No. 1, pp. 2-13.

[7] Oommen, B. J., Valiveti, R. S., and Zgierski, J. R. (1991). An Adaptive Learning Solution to the Keyboard Optimization Problem, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 21, No. 6, pp. 1608-1618.

[8] Oommen, B. J. and Croix, E. V. de St. (1996). Graph Partitioning Using Learning Automata, IEEE Trans. on Computers, No. 45, No. 2, pp. 195-208.

[9] Meybodi. M. R. and Lakshmivarhan, S. (1983). A Learning Approach to Priority Assignment in a Two Class M/M/1 Queuing System with Unknown Parameters, Proc. of Third Yale Workshop on Applications of Adaptive Systems Theory, Yale University, pp. 106-109.

[10] Narendra, K. S. and Thathachar, M. A. L. (1989). Learning Automata: An Introduction, Prentice-hall, Englewood cliffs.

[11] Thathachar, M. A. L. and Sastry, P. S. (1987). Learning Optimal Discriminant Functions Thought a Cooperative Game of Automata, IEEE Trans. Syst., Man and Cybern., Vol. SMC-27, pp.73-85.

[12] Mars, P., Chen, J. R., and Nambiar, R. (1998). Learning Algorithms: Theory and Application in Signal Processing, Control, and Communications, CRC press, New York.

[13] Mars. P. and Narendra. K. S., and Chrystall, M. (1983). Learning Automata Control of Computer Communication Networks, Proc. of Third Yale Workshop on Applications of Adaptive Systems Theory, Yale University.

[14] Meybodi, M. R. and Beigy, H. (2002) New Class of Learning Automata Based Scheme for Adaptation of Backpropagation Algorithm Parameters, International Journal of Neural Systems, vol. 12, No. 1, pp. 45-67.

[15] Beigy, H. and Meybodi, M. R. (2000). Solving the Graph Isomorphism Problem Using Learning Automata, In Proceedings of 5th Annual International Computer Society of Iran Computer Conference, CISCC-2000, pp. 402-415.

[16] Beigy, H. and Meybodi, M. R. (2001). Backpropagation Algorithm Adaptation Parameters using Learning Automata, International Journal of Neural Systems, Vol. 11, No. 3, pp. 219—228.

[17] Meybodi, M. R. and Beigy, H., “A Note on Learning Automata Based Schemes for Adaptation of BP Parameters”, Journal of Neurocomputing, accepted for publication.

[18] Hashim, A. A., Amir, S., and Mars, p. (1986). Application of Learning Automata to Data Compression, In Adaptive and Learning Systems, K. S. Narendra (Ed.), New York: Plenum Press, pp. 229-234.