A Nonlinear Reinforcement Scheme for Stochastic Learning Automata

FLORIN STOICA, EMIL M. POPA
Computer Science Department
University "Lucian Blaga" Sibiu
Str. Dr. Ion Ratiu 5-7, 550012, Sibiu
ROMANIA

Abstract: - A stochastic automaton can perform a finite number of actions in a random environment. When a specific action is performed, the environment responds by producing an environment output that is stochastically related to the action. This response may be favorable or unfavorable. The aim is to design an automaton that can determine the best action guided by past actions and responses. The reinforcement scheme presented is shown to satisfy all necessary and sufficient conditions for absolute expediency for a stationary environment. An automaton using this scheme is guaranteed to „do better" at every time step than at the previous step (expected value of the average penalty at one iteration step is less than of the previous step for all steps). Some simulation results are presented, which prove that our algorithm converges to a solution faster than the one given in [7].

Key-Words: - Stochastic Learning Automata, Reinforcement Learning

1 Introduction

An *automaton* is a machine or control mechanism designed to automatically follow a predetermined sequence of operations or respond to encoded instructions. The term *stochastic* emphasizes the adaptive nature of the automaton we describe here. The automaton described here do not follow predetermined rules, but adapts to changes in its environment. This adaptation is the result of the *learning* process. Learning is defined as any permanent change in behavior as a result of past experience, and a learning system should therefore have the ability to improve its behavior with time, toward a final goal.

The stochastic automaton attempts a solution of the problem without any information on the optimal action (initially, equal probabilities are attached to all the actions). One action is selected at random, the response from the environment is observed, action probabilities are updated based on that response, and the procedure is repeated. A stochastic automaton acting as described to improve its performance is called a *learning automaton*. The algorithm that guarantees the desired learning process is called a *reinforcement scheme* [5].

Mathematically, the environment is defined by a triple $\{a, c, b\}$ where $a = \{a_1, a_2, ..., a_r\}$ represents a finite set of actions being the input to the environment, $b = \{b_1, b_2\}$ represents a binary response set, and $c = \{c_1, c_2, ..., c_r\}$ is a set of penalty probabilities, where c_i is the probability that action a_i will result in an unfavourable response. Given that $b(n) = 0$ is a favourable outcome and $b(n) = 1$ is an unfavourable outcome at time instant n $(n = 0, 1, 2, ...)$, the element c_i of c is defined mathematically by:

$c_i = P(b(n) = 1 \mid a(n) = a_i) \quad i = 1, 2, ..., r$

The response values can be represented in three different models. In the P-model (described above), the response values are either 0 or 1, in the S-model the response values is continuous in the range (0, 1) and in the Q-model the values is in a finite set of discrete values in the range (0, 1).

The environment can further be split up in two types, stationary and nonstationary. In a stationary environment the penalty probabilities will never change. In a nonstationary environment the penalties will change over time.

2 Formal definition of a Stochastic Learning Automaton

The automaton is defined by a quintuple $\{\Phi, a, b, F(\bullet,\bullet), H(\bullet,\bullet)\}$.

$\Phi = \{\Phi_1, \Phi_2, ..., \Phi_s\}$ is the set of internal states of the automaton. The internal states determine the action to be produced by the automaton. The state at time instant n is denoted by $\Phi(n)$ and is an element of the finite set Φ.

$a = \{a_1, a_2, ..., a_r\}$ denotes the set of actions that can be produced by the automaton. This is the output set of the automaton, hence also being the input set to the environment. The action done at time instant n is denoted $a(n)$ and is an element of the finite set a.

$b = \{b_1, b_2, ..., b_m\}$ or $b = \{(a,b)\}$ is the input set to the automaton, that is the set of response from the environment. This set can be either finite or infinite. $b(n)$ denotes the input to the automaton at time instant n.

$F(\bullet,\bullet)$ is a function that maps the current state and the response from the environment into the next state, given mathematically by $F(\bullet,\bullet): \Phi \times b \to \Phi$. This formula is called the transition function. A similar expression is showed by the formula $\Phi(n+1) = F[\Phi(n), b(n)]$.

The transition function can be either deterministic or stochastic. If the function is deterministic the result of the function is uniquely specified for each state.

If the transition function F is stochastic, the elements f_{ij}^{b} of F represent the probability that the automaton moves from state Φ_i to state Φ_j given the input b :

$f_{ij}^{b} = P(\Phi(n+1) = \Phi_j \mid \Phi(n) = \Phi_i, b(n) = b)$

$i, j = 1, 2, ..., s$ and $b \in \{b_1, b_2, ..., b_m\}$

$H(\bullet,\bullet)$ is the output function and is defined mathematically by $H(\bullet,\bullet): \Phi \times b \to a$. This function maps the current state and the response from the environment into the action produced by the automaton. If the current output depends on only the current state, the automaton is referred to as *state-output automaton*. In this case the function $H(\bullet,\bullet)$ is replaced by an output function $G(\bullet): \Phi \to a$ which can be either deterministic or stochastic: $a(n) = G[\Phi(n)]$.

If G is stochastic, the elements of this set are denoted g_{ij}. The value of this element represents the probability that the action done by the automaton is a_j given the automaton is in state Φ_i:

$g_{ij} = P(a(n) = a_j \mid \Phi(n) = \Phi_i) \;\; i = \overline{1,s} \;\; j = \overline{1,r}$

In short the automaton takes an input from the environment and produces an action based on this. The automaton is showed in Figure 1:

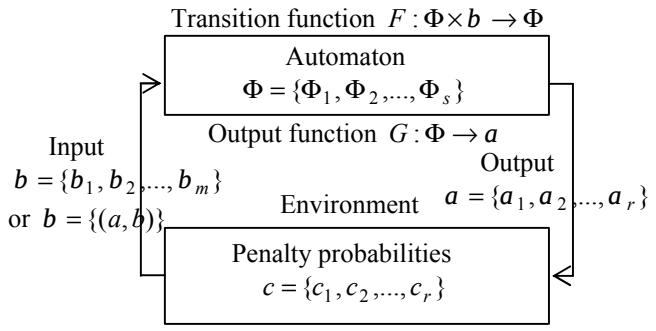

Fig. 1 A stochastic automaton

An automaton is called to be fixed structured when the functions f_{ij} and g_{ij} are having values that do not change over time. By making them change over time, one can get a greater flexibility where actions rewarded will get a higher chance of being chosen again. Such an automaton is called *variable-structure automaton*. Furthermore, in the case of variable-structure automaton, the above definitions of the transition functions F and G are not used explicitly. In order to describe the reinforcement schemes, is defined $p(n)$, a vector of action probabilities: $p_i(n) = P(a(n) = a_i), \; i = \overline{1,r}$

Updating action probabilities can be represented as follows:

$$p(n+1) = T[p(n), a(n), b(n)]$$

where T is a mapping. This formula says the next action probability $p(n+1)$ is updated based on the current probability $p(n)$, the input from the environment and the resulting action. If $p(n+1)$ is a linear function of $p(n)$, the reinforcement scheme is said to be linear; otherwise it is termed nonlinear.

3 Variable Structure Automaton

3.1 Performance Evaluation

A learning automaton generates a sequence of actions on the basis of its interaction with the environment. If the automaton is "learning" in the process, its performance must be superior to "intuitive" methods. In the following we will consider the simplest case, the P-model and stationary random environments.

Consider a stationary random environment with penalty probabilities

$\{c_1, c_2, ..., c_r\}$ where $c_i = P(b(n) = 1 \mid a(n) = a_i)$.

We define a quantity $M(n)$ as the average penalty for a given action probability vector:

$M(n) = P(b(n) = 1 \mid p(n)) =$

$$\sum_{i=1}^{r} P(b(n) = 1 \mid a(n) = a_i) * P(a(n) = a_i) = \sum_{i=1}^{r} c_i p_i(n)$$

An automaton is absolutely expedient if the expected value of the average penalty at one iteration step is less than it was at the previous step *for all steps*: $M(n+1) < M(n)$ for all n [8].

Absolutely expedient learning schemes are presently the only class of schemes for which necessary and sufficient conditions of design are available. The algorithm we will present in this paper is derived from a nonlinear absolutely expedient reinforcement scheme presented by [7].

3.2 The general reinforcement scheme

Consider a variable-structure automaton with *r* actions in a stationary environment with $b = \{0,1\}$. The general scheme for updating action probabilities is as follows:

If $a(n) = a_i$, when $b = 0$

$p_j(n+1) = p_j(n) - g_j(p(n))$ for all $j \neq i$ and

$$p_i(n+1) = p_i(n) + \sum_{\substack{k=1 \\ k \neq i}}^{r} g_k(p(n)). \quad (1)$$

When $b = 1$, $p_j(n+1) = p_j(n) + h_j(p(n))$ for all $j \neq i$ and $p_i(n+1) = p_i(n) - \sum_{\substack{k=1 \\ k \neq i}}^{r} h_k(p(n))$ where g_k and $h_k, k = 1,2,...,r$ are continuous, nonnegative functions with the following assumptions:

$$0 < g_k(p(n)) < p_k(n)$$

$$0 < \sum_{\substack{k=1 \\ k \neq i}}^{r} [p_k(n) + h_k(p(n))] < 1$$

for all $i = 1,2,...,r$ and all probabilities p_k in the open interval (0, 1).

3.3 Absolutely expedient reinforcement schemes

The reinforcement scheme is the basis of the learning process for learning automata. The general solution for absolutely expedient schemes was found by Lakshmivarahan and Thathachar [11]. Consider the general reinforcement scheme in equations (1). A learning automaton using this scheme is absolutely expedient if and only if the functions $g_k(p(n))$ and $h_k(p(n))$ satisfy the following conditions:

$$\frac{g_1(p(n))}{p_1(n)} = \frac{g_2(p(n))}{p_2(n)} = ... = \frac{g_r(p(n))}{p_r(n)} = l(p(n))$$

$$\frac{h_1(p(n))}{p_1(n)} = \frac{h_2(p(n))}{p_2(n)} = ... = \frac{h_r(p(n))}{p_r(n)} = m(p(n))$$

where $l(p(n))$ and $m(p(n))$ are arbitrary functions satisfying:

$$0 < l(p(n)) < 1$$

$$0 < m(p(n)) < \min_{j=1,...,r} \left\{ \frac{p_j(n)}{1 - p_j(n)} \right\}$$

Detailed proof of this theorem is given in [8].

A learning automaton may send its action to multiple environments at the same time. In that case, the action of the automaton results in a vector of responses from environments (or "teachers"). In a stationary N-teacher P-model environment, if an automaton produced the action a_i and the environment responses are $b_i^j \; j = 1,...,N$ at time instant n, then the vector of action probabilities $p(n)$ is updated as follows [7]:

$$p_i(n+1) = p_i(n) + \left[\frac{1}{N} \sum_{k=1}^{N} b_i^k \right] * \sum_{\substack{j=1 \\ j \neq i}}^{r} f_j(p(n)) -$$

$$- \left[1 - \frac{1}{N} \sum_{k=1}^{N} b_i^k \right] * \sum_{\substack{j=1 \\ j \neq i}}^{r} y_j(p(n)) \quad (2)$$

$$p_j(n+1) = p_j(n) - \left[\frac{1}{N} \sum_{k=1}^{N} b_i^k \right] * f_j(p(n)) +$$

$$+ \left[1 - \frac{1}{N} \sum_{k=1}^{N} b_i^k \right] * y_j(p(n))$$

for all $j \neq i$ where the functions f_i and y_i satisfy the following conditions:

$$\frac{f_1(p(n))}{p_1(n)} = ... = \frac{f_r(p(n))}{p_r(n)} = l(p(n)) \quad (3)$$

$$\frac{y_1(p(n))}{p_1(n)} = ... = \frac{y_r(p(n))}{p_r(n)} = m(p(n))$$

$$p_i(n) + \sum_{\substack{j=1 \\ j \neq i}}^{r} f_j(p(n)) > 0 \quad (4)$$

$$p_i(n) - \sum_{\substack{j=1 \\ j \neq i}}^{r} y_j(p(n)) < 1 \quad (5)$$

$$p_j(n) + y_j(p(n)) > 0 \quad (6)$$

$$p_j(n) - f_j(p(n)) < 1 \quad (7)$$

for all $j \in \{1,...,r\} \setminus \{i\}$

The conditions (4)-(7) ensure that $0 < p_k < 1, \; k = \overline{1,r}$.

Theorem If the functions $l(p(n))$ and $m(p(n))$ satisfy the following conditions:

$$l(p(n)) \leq 0$$

$$m(p(n)) \leq 0 \quad (8)$$

$$l(p(n)) + m(p(n)) < 0$$

then the automaton with the reinforcement scheme in (2) is absolutely expedient in a stationary environment.

The proof of this theorem can be found in [9].

4 A new nonlinear reinforcement scheme

Because the above theorem is also valid for a single-teacher model, we can define a single environment response that is a function f of many teacher outputs.

Thus, we can update the above algorithm as follows:

$$\begin{aligned} &p_i(n+1) = p_i(n) + f*(-q*H(n))*[1-p_i(n)] - \\ &-(1-f)*(-q)*[1-p_i(n)] \\ &p_j(n+1) = p_j(n) - f*(-q*H(n))*p_j(n) + \\ &+(1-f)*(-q)*p_j(n) \end{aligned} \quad (9)$$

for all $j \neq i$, i.e.:

$$y_k(p(n)) = -q*p_k(n)$$
$$f_k(p(n)) = -q*H(n)*p_k(n)$$

where the learning parameter q is a real value which satisfy: $0<q<1$.

The function H is defined as:

$$H(n) = \min\left\{1; \max\left\{\min\left\{\frac{p_i(n)}{q(1-p_i(n))} - e, \left(\frac{1-p_j(n)}{q*p_j(n)} - e\right)_{\substack{j=\overline{1,r} \\ j\neq i}}\right\}; 0\right\}\right\}$$

Parameter e is an arbitrarily small positive real number.

Our reinforcement scheme differs from the one given in [6], [7] by the definition of these two functions: H and f_k.

We now show that are satisfied all the conditions of the reinforcement scheme.

From (3) we have:

$$\frac{f_k(p(n))}{p_k(n)} = \frac{-q*H(n)*p_k(n)}{p_k(n)} = -q*H(n) = l(p(n))$$

$$\frac{y_k(p(n))}{p_k(n)} = \frac{-q*p_k(n)}{p_k(n)} = -q = m(p(n))$$

The rest of the conditions translates to the following:

Condition (4):

$$p_i(n) + \sum_{\substack{j=1 \\ j\neq i}}^{r} f_j(p(n)) > 0 \Leftrightarrow$$

$$p_i(n) - q*H(n)*(1-p_i(n)) > 0 \Leftrightarrow$$

$$q*H(n)*(1-p_i(n)) < p_i(n) \Leftrightarrow H(n) < \frac{p_i(n)}{q*(1-p_i(n))}$$

This condition is satisfied by the definition of the function $H(n)$.

Condition (5):

$$p_i(n) - \sum_{\substack{j=1 \\ j\neq i}}^{r} y_j(p(n)) < 1 \Leftrightarrow p_i(n) + q*(1-p_i(n)) < 1$$

But $p_i(n) + q*(1-p_i(n)) < p_i(n) + 1 - p_i(n) = 1$ since $0<q<1$

Condition (6):

$p_j(n) + y_j(p(n)) > 0 \Leftrightarrow p_j(n) - q*p_j(n) > 0$ for all $j \in \{1,...,r\} \setminus \{i\}$

But $p_j(n) - q*p_j(n) = p_j(n)*(1-q) > 0$ since $0<q<1$ and $0<p_j(n)<1$ for all $j \in \{1,...,r\} \setminus \{i\}$

Condition (7):

$p_j(n) - f_j(p(n)) < 1 \Leftrightarrow p_j(n) + q*H(n)*p_j(n) < 1$ for all $j \in \{1,...,r\} \setminus \{i\}$

We have:

$$p_j(n) + q*H(n)*p_j(n) < 1 \Leftrightarrow H(n) < \frac{1-p_j(n)}{q*p_j(n)}$$ for all $j \in \{1,...,r\} \setminus \{i\}$.

This condition is satisfied by the definition of the function $H(n)$.

With all conditions of the equations (2) satisfied, we conclude that the reinforcement scheme is a candidate for absolute expediency.

Furthermore, the functions l and m for our nonlinear scheme satisfy the following:

$$l(p(n)) = -q*H(n) \leq 0$$
$$m(p(n)) = -q \leq 0$$
$$l(p(n)) + m(p(n)) = -q*(1+H(n)) < 0$$

because $0<q<1$ and $0 \leq H(n) \leq 1$

In conclusion, we state the algorithm given in equations (9) is absolutely expedient in a stationary environment.

5 Simulation results

5.1 Problem formulation

Reinforcement learning is justified if it is easier to implement the reinforcement function than the desired behavior, or if the behavior generated by the agent presents desirable emergent properties (like generalization, robustness, redundancy, adaptability) which cannot be directly built. This last reason is certainly the best motivation for the use of reinforcement learning in autonomous robotics.

To show that our algorithm converges to a solution faster thean the one given in [7], let us consider a simple example. Figure 2 illustrates a grid world in which a robot navigates. Shaded cells represent barriers.

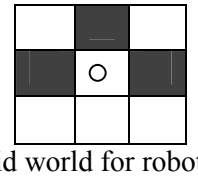

Fig. 2 A grid world for robot navigation

The current position of the robot is marked by a circle. Navigation is done using four actions $a = \{N, S, E, W\}$,

the actions denoting the four possible movements along the coordinate directions.

5.2 Implementation of the learning process

The algorithm used is:

Step 1. Choose an action, $a(n) = a_i$ based on the action probability vector $p(n)$.

Step 2. Read environment responses b_i^j, $j = \overline{1,4}$ from the sensor modules.

Step 3. Compute the combined environment responses *f*.

Step 4. Update action probabilities $p(n)$ according to the predefined reinforcement scheme.

Step 5. Go to step 1.

Because in given situation there is a single optimal action, we stop the execution when the probability of the optimal action reaches a certain value (0.9999).

The Java class which implements the stochastic automaton is:

```
public class Automaton {
   // vector of action probabilities
   double p[] = new double[4];
   // used to choose an action according to
   // the vector of action probabilities
   double F[] = new double[5];
   // learning parameter
   double t = 0.02;
   // small positive real number used in
   // definition of function H
   double eps = 0.00001;
   // the set of actions
   static final int N = 0;
   static final int S = 1; // optimal action
   static final int E = 2;
   static final int W = 3;

   public void init(boolean equal){
     if (equal)
        // all probabilities are initially the same
        for (int i=0; i < 4; i++) p[i]=0.25;
     else {
        for (int i=0; i < 4; i++) p[i]=0.9995/3;
        // the optimal action initially has a small
        // probability value
        p[S]=0.0005;
     }
   }
   // choose an action, based on the action
   // probability vector
   public int getAction(){
     F[0]=0;
     for (int i=1; i<=4;i++)
         F[i]=F[i-1]+p[i-1];
     double nr = Math.random();
     int choice = -1;
     for (int i = 0; i<=4; i++)
         if (F[i] > nr) {
           choice = i-1;
           break;
         }
     return choice;
   }
   // environment response
   public double reward(int action){
     if (action != S)
        return 1;  // unfavourable outcome
     else
        return 0;  // favourable outcome
   }                // for the optimal action
   // function H from reinforcement scheme
   public double H(int i){
     double h, temp;
     h = p[i]/(t*(1-p[i]))-eps;
     for (int j=0; j < 4; j++)
        if (j!=i) {
           temp=(1-p[j])/(t*p[j])-eps;
           h = Math.min(h,temp);
        }
     h = Math.max(h,0);
     h = Math.min(h,1);
     return h;
   }
   // learning process
   public int learning(boolean equal){
     int i, j, k=0;
     double f, h;
     init(equal);
     while (true){
       k++;
       // choose an action
       i = getAction();
       // compute environment response
       f = reward(i);
       h = H(i);
       // update action probabilities
       // according to the our reinforcement scheme
       p[i]=p[i]+f*(-t*h)*(1.0-p[i])-(1.0-f)*(-t)*(1.0-p[i]);
       for (j=0; j < 4; j++)
           if (j != i)
              p[j]=p[j]-f*(-t*h)*p[j]+(1.0-f)*(-t)*p[j];
       // stop the execution when the probability of the
       // optimal action (S) reaches a certain value
       if (p[S] > 0.9999) {
          return k;
       }
     }
   }
}
```

5.3 Comparative results

We compared two reinforcement schemes using four actions and two different initial conditions. The data shown in Table 1 are the results of two different initial conditions where in first case all probabilities are initially the same and in second case the optimal action initially has a small probability value (0.0005), with only one action receiving reward (i.e., optimal action).

	Average number of steps to reach p_{opt}=0.9999			
	4 actions with $p_i(0) = 1/4, i = \overline{1,4}$		4 actions with $p_{opt}(0) = 0.0005$, $p_{i \neq opt} = 0.9995/3$	
q	Alg. From [7]	New alg.	Alg. From [7]	New alg.
0.01	644.835	633.960	921.200	905.180
0.03	222.445	216.685	332.890	324.590
0.05	136.985	130.405	202.955	198.245
0.1	70.805	63.085	105.115	99.200
0.2	34.730	30.900	53.345	50.930
0.5	11.055	9.750	22.500	19.430

Table 1 Convergence rates for a single optimal action of a 4-action automaton in a stationary environment (200 runs for each parameter set)

Comparing values from corresponding columns, we conclude that our algorithm converges to a solution faster than the one given in [7].

6 Conclusion

Reinforcement learning has attracted rapidly increasing interest in the machine learning and artificial intelligence communities. Its promise is beguiling - a way of programming agents by reward and punishment without needing to specify how the task (i.e., behavior) is to be achieved. Reinforcement learning allows, at least in principle, to bypass the problems of building an explicit model of the behavior to be synthesized and its counterpart, a meaningful learning base (supervised learning).

The reinforcement scheme presented in this paper satisfy all necessary and sufficient conditions for absolute expediency in a stationary environment, and the nonlinear algorithm based on this scheme is found to converge to the „optimal" action faster than nonlinear schemes previously defined in [9], [7].

References:

[1] A. Barto, S. Mahadevan, Recent advances in hierarchical reinforcement learning, *Discrete-Event Systems journal, Special issue on Reinforcement Learning*, 2003.

[2] R. Sutton, A. Barto, *Reinforcement learning: An introduction*, MIT-press, Cambridge, MA, 1998.

[3] O. Buffet, A. Dutech, and F. Charpillet. Incremental reinforcement learning for designing multi-agent systems, In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference onAutonomous Agents*, pp. 31–32,Montreal, Canada, 2001. ACM Press.

[4] J. Moody, Y. Liu, M. Saffell, and K. Youn. Stochastic direct reinforcement: Application to simple games with recurrence, In *Proceedings of Artificial Multiagent Learning. Papers from the 2004 AAAI Fall Symposium,Technical Report FS-04-02*, 2004.

[5] Cem Ünsal, Pushkin Kachroo, John S. Bay, Simulation Study of Learning Automata Games in Automated Highway Systems, *1st IEEE Conference on Intelligent Transportation Systems (ITSC'97)*, Boston, Massachusetts, Nov. 9-12, 1997

[6] Cem Ünsal, Pushkin Kachroo, John S. Bay, Simulation Study of Multiple Intelligent Vehicle Control using Stochastic Learning Automata, *TRANSACTIONS, the Quarterly Archival Journal of the Society for Computer Simulation International*, volume 14, number 4, December 1997.

[7] Cem Ünsal, Pushkin Kachroo, John S. Bay, Multiple Stochastic Learning Automata for Vehicle Path Control in an Automated Highway System, *IEEE Transactions on Systems, Man, and Cybernetics -part A: systems and humans*, vol. 29, no. 1, january 1999

[8] K. S. Narendra, M. A. L. Thathachar, *Learning Automata: an introduction*, Prentice-Hall, 1989.

[9]N. Baba, New Topics in Learning Automata: Theory and Applications, *Lecture Notes in Control and Information Sciences* Berlin, Germany: Springer-Verlag, 1984.

[10] Dorigo, M., Introduction to the Special Issue on Learning Autonomous Robots, *IEEE Trans. on Systems, Man and Cybernetics - part B*, Vol. 26, No. 3, 361-364, 1996.

[11] S. Lakshmivarahan, M.A.L. Thathachar, Absolutely Expedient Learning Algorithms for Stochastic Automata, *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-6, pp. 281-286, 1973

[12] C. Rivero, Characterization of the absolutely expedient learning algorithms for stochastic automata in a non-discrete space of actions, *ESANN'2003 proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium)*, 23-25 April 2003, d-side publi., ISBN 2-930307-03-X, pp. 307-312

[13] K.P. Topon, I. Hitoshi, Reinforcement Learning Estimation of Distribution Algorithm, *Proceedings of the Genetic and Evolutionary Computation Conference 2003 (GECCO2003)*