Paper

Learning Cellular Automata for Function Optimization Problems

Member Fei Qian (Hiroshima Kokusai Gakuin University)
Member Yue Zhao (Hiroshima Kokusai Gakuin University)
Member Hironori Hirata (Chiba University)

We present a model of learning cellular automata (LCA) as an emergent system having some collective behaviors. LCA is an extended version of the traditional cellular automaton. Especially, we adopt the LCA with some self-improving functions, called self-improving learning cellular automata (SILCA) and develop its optimization capability. Each self-improving learning cellular automaton, i.e. a member of SILCA, consists of two parts: *main body* and *a universal constructor*. Through the use of a *constructing arm*, a learning cellular automaton is capable of constructing any configuration whose description can be stored on its input tape. As an example of combinatorial optimization problems, we consider function optimization problems and show the SILCA's emergent capability for optimization.

Keywords: learning automata, cellular automata, function optimization problems

1. Introduction

In the complex adaptive systems, interesting global behaviors emerge from many local interactions. When such emergent behavior is a computation, we refer to it as an emergent computation. The emergent computation should be constructed by exploiting interactions among primitive components. The research on self-improving or self-reproducing behavior may play an important role in the emergent computation.

In the self-reproducing cellular automata of von Neumann [1], the copy is created by interpreting the coded description on the tape as a sequence of instructions that cause the construction, cell by cell, of a new machine in an unoccupied part of cellular space. The copying actually occurs at the moment when the state of some previously quiescent cells immediately adjacent to the tip of the constructing arm is changed to the desired new state. To evaluate the copy, the free energy must be supplied to the system to enable it to change states. The force that causes the change of state corresponds to the free energy of the system.

In particular, von Neumann demonstrated the possibility of self-replicatability of a computational structure by actually designing a self-reproducing automaton consisting of a two-dimensional cellular arrangement with a large number of individual 29-state cells. The next state of the 29-state automaton is a function of its own current state and the state of its four neighbors in the two-dimensional cellular space. Within this framework, von Neumann was able to conceive a self-reproducing automaton endowed with the properties of both computational and constructional universality. Unfortunately, the automaton was of such complexity that notwithstanding further simplifications, even today's state-of-the-art computers lack the power to simulate it in its entirety.

Here, we present a model learning cellular automata (LCA) that is an extended version of the traditional cellular automata, with some self-improving functions, called SILCA. We aim at using this model to solve some combinatorial optimization problems in some environments with non-complete prior information (e.g. a traverling selseman problem (TSP) with no size information, etc.).

The remainder of this paper is organized as follows. Section 2 describes the definitions of learning cellular automata. This is followed by Section 3, that deals with the element presentation and the reinforcement schemes. In Section 4 the structure of the SILCA is shown. In the Section 5, the design of main body of SILCA for some function optimization problems is presented.

2. Definitions of learning cellular automata

Traditional stochastic cellular automata are just as computationally capable as the production system that underlies a classifier system. Therefore one would expect the stochastic cellular automata to be just as hard to find the global optimum as the classifier systems in that: 1) the former relies almost entirely on a large number of extremely simple local rules to perform powerful computations collectively, whereas the latter typically re-quires much more complex rules to execute the same tasks; 2) even though the stochastic cellular auto-mata and production systems employ discrete set of rules, the stochastic nature of the state transitions and input-output relations allows the stochastic cellular automata to be trained using continuous optimization techniques in addition to discrete ones. Both of these promise a drastic reduction of learning complexity.

The approach we take in implementing the reinforcement learning in stochastic cellular automata with random environment is closely parallel to that of stochastic learning automata [2] [3]. The basic idea of our approach is to modify either the state transition probabil-

ities or the probabilities of the learning automaton in order to appropriately reflect the random environmental response, such as rewards or penalties which represent successable or unsuccessable reactions of the environment to the system response.

A stochastic learning automaton is an abstract machine consisting of a stochastic automaton and its environment together in a feedback loop. Typically, it is assumed that there are finitely many possible actions for the automaton. In this case, let $\boldsymbol{Y} = \{y_1, \cdots, \boldsymbol{y}_n\}$ be the set of possible actions of the automaton. A probability distribution over these actions is then a set of numbers $\boldsymbol{p} = \{p_1, \cdots, p_n\}$, where p_j represents the probability of choosing actions y_j. Furthermore, $\sum_{j=1}^{n} p_j = 1$ and $p_j \geq 0$ for all j. Thus such a distribution can be identified with a $(n-1)$-dimensional simplex. The state of the learning automaton at any time can thus be thought of as a point in the simplex of appropriate dimension. The coordinates of $\boldsymbol{p}$ are called the *action probabilities*.

The evaluative feedback provided by the environment is called the reinforcement signal. It is often assumed that the reinforcement has just two possible values, *success* or *failure*. In this case the probability distribution of reinforcement corresponding to action y_j is specified by a number c_j, which represents the probability that reinforcement is success when the automaton has chosen action y_j. The numbers $\{c_1, \cdots, c_n\}$ are called the *penalty probabilities*.

Definition 1 *A learning cellular automata (LCA) A and its environment E are defined as follows.*

$$A = \{U, X, Y, Q, N, \xi, F, O, T\} \quad \cdots\cdots\cdots\cdots (1)$$

$$E = \{Y, C, r\} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2)$$

where,

U : *The cellular space.*$U = \{u_j, j = 1, 2, \ldots, n\}$
X : *The set of inputs.*$X = \{x_j, 0 \leq j < \infty\}$
Y : *The set of outputs.*$Y = \{y_j, 0 \leq j < \infty\}$
N : *The list of neighborhood relations.* $N = \{n^1, \cdots, n^{|N|}\}$
Q : *The set of internal states.* $Q = \{q_j, 0 \leq j < \infty\}$
ξ : *The neighborhood state configuration function.* $\xi : U \to \Omega, \Omega \subseteq U$
F : *The stochastic state transition function.* $F : Q \times X \times r \to Q$
O : *The stochastic output function.*$O : Q \to Y$
T : *The reinforcement scheme.*$Q(t+1) = T(Q(t))$
C : *The penalty probability distribution.* $C = \{c_j, 0 \leq j < \infty\}$
r : *The reinforcement signal.*$r = \{r_j, 0 \leq j < \infty\}$

Remark 1 For all $n^i \in N$, n^i determines the relative positions of the neighboring sites from any given site i. Call n^i as nearest neighbors of i , if the Hamming distance is one between n^{i_j} $(n^{i_j} \in n^i)$ and i.

Remark 2 The neighborhood state configuration function ξ maps any given site i as specified by the neighborhood relationship defined by N.

$$\xi : Q \to \bigotimes_{n^i \in N} q_{n^i} \ , \quad q_{n^i} \stackrel{\Delta}{=} \left(q_{n^i,1}, \cdots, q_{n^i,|n^i|}\right) \quad (3)$$

Remark 3 The stochastic state transition function F maps the current input x_i and neighborhood configuration n^i into the next state q at any given site with a probability $\eta(x_i, n^i, q)$.

$$\forall x_i \in X, \forall n^i \in N \ : \ \sum_{q \in Q} \eta(x_i, n^i, q) = 1 \ \cdots\cdots (4)$$

Remark 4 The stochastic output function O maps the neighborhood configuration n^i into the output y_i at any given site i with a probability $\zeta(n^i, y_i)$.

$$\forall n^i \in N \ : \ \sum_{y_i \in Y} \zeta(n^i, y_i) = 1 \ \cdots\cdots\cdots\cdots\cdots (5)$$

Here, the O is assumed as a deterministic mapping, and for all $y \in Y, y(t) \stackrel{\Delta}{=} \zeta(Q(T))$ mapped as one to one.

Remark 5 The reinforcement scheme T is usually given as an algorithm which updates the current $\zeta(t)$ to $\zeta(t+1)$

Remark 6 The random environment E is assumed stationary and is characterized by the penalty probability distribution C. For all $c^i \in C$, c^i is a set of probabilities with its elements:

$$c_j^i = prob\left[r = -1 | y^i = y_j^i, x^i\right] \ \cdots\cdots\cdots\cdots (6)$$

$$for\ j = 1, 2, \cdots, |n^i|$$

where reinforcement signal $r = -1$ is associated with penalty and $r = +1$ with reward, and c_j^i is the probability that a given input-output pair will receive penalty from the environment.

3. The representation of elements and the reinforcement schemes

There are two directions to generate the notion of learning automata for the purpose of studying their behavior as units in connectionist networks. Certain special cases of these generalizations are already implicit in the work of Williams [4] and Barto [5]. Here we make them explicit and reduce them to the elements of the stochastic learning cellular automata.

For a learning automaton having $n+1$ actions, its state space Q can be represented as the n-dimensional simplexΔ^n. One natural generation is to allow Q to be arbitrary and assume a given mapping $\eta : Q \to \Delta^n$. For a two-action automaton, the state is obtained by letting Q be the real space and letting $\eta : Q \to \Delta^1$ be determined by some one to one mapping from Q to the $(0, 1)$, we let mapping $p_1 = \eta(q)$ represent the probability of selecting action y_1 when the automaton is in the state q. The state variable q is defined as a pair of input and neighborhood configuration to the automaton.

For an n-action automaton, an example of such a mapping is obtained as follows. Consider a random variable with parameters $\tilde{p}$ and n. The idea is to let

the probability that y_i is selected equal the probability $\eta(\tilde{p})$. We can set $\eta(\tilde{p})$ to the binomial probabilities $C_i^n \tilde{p}(1-\tilde{p}_i)^{n-i}$, where the parameter $\tilde{p}_i$ depends on the state of automaton $j(j \in n^i)$ and the synaptic strengths from n^i to i.

For such an automaton, let us consider the following dynamics:

$$E\big[\Delta w_{ij}|w\big] = \kappa \frac{\partial E[r|w]}{\partial w_{ij}}, \quad j \in n^i \quad \cdots\cdots\cdots (7)$$

where w_{ij} are often called *synaptic weights* between site i and j, and w called the *weight space*.

Equation (7) meaning that we want to control the gradient of $E[r|w]$ in weight space of the expected reinforcement, and to find a learning algorithm for updating the state transition probabilities by updating the weight strength.

From some simple computations, the following results can be obtained.

$$\Delta w_{ij} = \kappa r(t) f'(\mu(t)) q_j(t) \frac{\partial}{\partial p_i} logh(p_i) \quad \cdots\cdots (8)$$

where $\kappa > 0$ is a learning parameter, $\mu(t) = \sum_{j \in n^i} w_{ij}(t) q_j(t)$. According to simplex constraint condition, $f(\cdot)$ is a squashing function increasing in $(0,1)$ continuously. For an example, we can set it as

$$f(\mu) = (1 + exp(\mu/T_0))^{-1} \quad \cdots\cdots\cdots\cdots\cdots (9)$$

where T_0 is a smoothing parameter.

There is a large number of possible learning algorithms for such stochastic state transition function. If we restrict attention to the reward/penalty reinforcement, the idea is to reward actions leading to success and to penalize actions leading to failure. Here, we consider a parameterized-state method and presented η as a binomial probability distribution. In such case, the $q_i(t)$ can be considered as a *Bernoulli* random variable with a parameter $\tilde{p}_i = f(\mu)$.

For such site i, we construct the local rule which updates the state transition probability p^i as follows.

Binomial type local rule T_b:
Parameters κ : the learning parameter.
Method
Initialize $p_j^i(0) = 0.5, \quad j = 1, 2$
Do

$$p^i(t+1) = \Big(1 - \eta(\tilde{p}_i(t)), \eta(\tilde{p}_i(t))\Big)^T$$

$$w_{ij}(t+1) = w_{ij}(t) + \kappa r(t)\Big(q_i(t) - \tilde{p}_i(t)\Big) q_j(t)$$

EndDo
EndMethod

Theorem 1 *The Binomial type local rule T_b is an absolutely expedient scheme.*

There are two important ways to implement the rule space: parallel and sequential. For the former we have the following iteration scheme:

$$q_i(t+1) = \eta_i\Big(\sum_{j=1}^{n} w_{ij} q_j(t)\Big) \quad \cdots\cdots\cdots\cdots (10)$$

and for the sequential model:

$$q_i(t+1) = \eta_i\Big(\sum_{j<i}^{n} w_{ij} q_j(t+1) + \sum_{j \geq i}^{n} w_{ij} q_j(t)\Big) \quad (11)$$

Definition 2 *An LCA is called uniform if the neighborhood relation and local state transition function are the same for every site.*

For this two models some convergence behaviors and applications have presented [6] [7].

4. The structure of the self-improving learning cellular automata

A self-improving learning cellular automaton (Fig.1), i.e., a member of SILCA, consists of two parts: the *main body* and the *universal constructor*. The learning cellular automaton is capable of constructing, through the use of a constructing arm, any configuration whose description can be stored on its input tape. Using this constructing arm, the automaton is able to construct a copy (an offspring) of itself. From the macroscopic viewpoint, this consideration is the same as *von Neumann*'s *self-reproducing automaton*, but from the microscopic viewpoint the structure and the function are just *self-improving*.

Definition 3 *The self-improving learning cellular automaton is defined as an stochastic cellular automaton A containing an universal constructor W, with its own random environment E. Here:*

$$A = \{U, X, Y, Q, N, \xi, F, O, T, W\} \quad \cdots\cdots\cdots (12)$$

$$E = \{Y, C, r\} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (13)$$

where,
U : *The cellular space.*$U = \{u_j, j = 1, 2, \ldots, n\}$
X : *The set of inputs.*$X = \{x_j, 0 \leq j < \infty\}$
Y : *The set of outputs.*$Y = \{y_j, 0 \leq j < \infty\}$
N : *The list of neighborhood relations.* $N = \{n^1, \cdots, n^{|N|}\}$
Q : *The set of internal states.* $Q = \{q_j, 0 \leq j < \infty\}$
ξ : *The neighborhood state configuration function.* $\xi : U \to \Omega, \Omega \subseteq U$
F : *The stochastic state transition function.* $F : Q \times X \times r \to Q$
O : *The stochastic output function.*$O : Q \to Y$
W : *The constructor.*$W = \{W_c, W_f\}$
T : *The reinforcement scheme.*$Q(t+1) = T(Q(t))$
C : *The penalty probability distribution.* $C = \{c_j, 0 \leq j < \infty\}$
r : *The reinforcement signal.*$r = \{r_j, 0 \leq j < \infty\}$

Here, we will confine our attention to S-model learning automaton [2] [3], i.e. for the case $r = \{r_j, 0 \leq j < \infty\}$, where the r_j represents the strength of the penalty responses.

For constructor W, W_c is the constructing state space

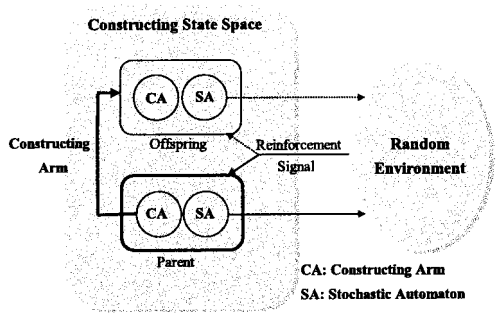


Fig. 1. The learning cellular automaton with self-improving

$W_c = \{s_{ij}, i, j = 0, 1, \ldots\}$, where the state s_{ij} is respectively defined as 0 for quiescent state; 1, sheath state; 2, activation state; 3, construction state; 4, destruction; d, date transmission state, and W_f is the function control unit.

To make an account of the constructing state space and the universal constructor, there are some representations have been provided by *von Neumann* [1], *Codd* [8], *Langton* [9] and other researchers.

Langton's "periodic emitter" [9] is based on Codd's universal constructors [8]. The constructor (automaton) (Fig. 2) is essentially a duplicated square loop, we call those as *internal sheath* and *external sheath* respectively. The states necessary for the construction of duplicate loop circulate counterclockwise. A reproduction is achieved by extending a constructing arm to the left at regular intervals corresponding to the size of one side of the loop. After such runs, the constructing arm will have folded upon itself. When the new constructing loop is closed, the constructing arm will be cut down and the new offspring with new constructing loop will be obtained. Finally, the parent loop will die when all offspring of itself in another direction have reproduced. The constructor will replicate itself to fill the available constructing state space, when one gives it sufficient time.

Here, our considerations are similar to those about Langton's loop. For each cell of constructor we use a 9-neighborhood (containing itself also) and use the internal sheath only. In our internal sheath, we have two (Fig.3.) or four (Fig.4.) out-going cells at the two (left-top and right-bottom) or four corners of the constructor. Initially, these two or four cells are in the open state, and will change to the closed state once the offspring is

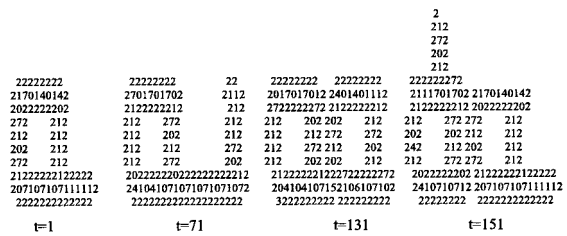


Fig. 2. Langton's periodic emitter

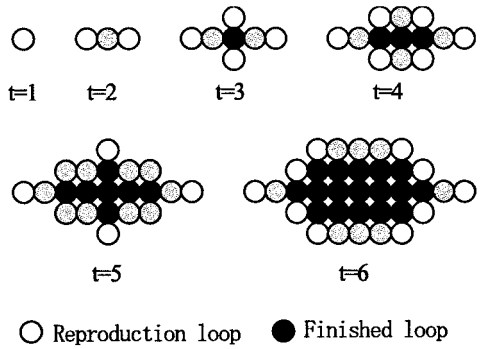


Fig. 3. The two out-going cells loop and its generations.

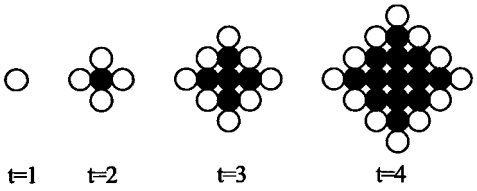


Fig. 4. The four out-going cells loop and its generations.

accomplished.

The initial configuration is in the form of a square loop wrapped around a sheath. The size of the loop is variable. We can use any size of loop, for example 8x8. Near the out-going cells of the loop, the initial state is in the sheath state 1. The automaton should attempt to duplicate itself in all out-going directions with the following rule.

Reproducing rule$R^{5,4}$ † :

(1) Once the constructor starts, operating the constructing date starts running around the loop.

(2) When the sheath state 2 (we call it as messenger), reaches a out-going cell and finds it open, the messenger splits into the same two cells. One cell whose left neighbor is in destruction state continues turning around the loop, and another starts extending the constructing arm.

(3) Once the constructing arm has start extending, each messenger that arrives to out-going cells will again split and one of the parent's copies will send though the arm.

(4) Once the first messenger reaches the tip of the constructing arm, the state of the tip changes to activation state (3). The constructing arm will start the offspring's sheath constructing.

(5) The next messengers will turn the tip of the constructing arm to the left, until the four corners have completed. The last messenger will cause the sheath to close and then change the state to activation state,

† The rule for 5-states and 4-out-going cells.

and turns back this state through the arm, as an response signal, to responds the parent loop that the offspring's sheath is ready.

(6) When the response signal form the offspring arrives the corner of parent's loop, and the next messenger from the parent's loop have arrived, the messenger will split itself again, one copy running along the constructing arm will carry a copy of the parent's data in the loop. We call this messenger a carrier.

(7) Once the copying of the parent's data has finished, the last carrier will change to destructor with destruction state (4). The destructor will run along the constructing arm to the parent and break down the constructing arm.

(8) When the destructor arrives at the corner of the parent's loop, the state of the loop will change to quiescent state (0), the constructing function of the parent will die.

5. Design of the main body for function optimization problems

To show how the self-reproducing learning cellular automata work, we try to solve the following optimization problems.

TASK 1: (Fig.5 (a))

$$\min_{x,y\in[-10,10]}\Big\{f(x,y)= 4-\frac{\cos(2\pi x+1)\cos(2\pi y+1)}{0.25(1+0.2x^2)(1+0.2y^2)}\Big\} \quad (14)$$

TASK 2: : (*Griewank function*, Fig.5 (b))

$$\min_{x,y\in[-10,10]}\Big\{f(x,y)= \frac{x^2+y^2}{50}-\cos x\cos\frac{y}{\sqrt{2}}+1\Big\} \quad \cdot \quad (15)$$

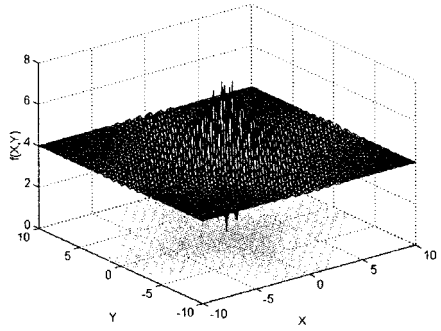
As shown in Fig. 5 (a),(b), these functions have so many local minima in the solution space, and get the global minimum 0 at $x = y = 0$. Because of being trapped in the local minima, traditional downhill methods will not work well to solve these problems. Therefore, we design a variable structure of hierarchical learning automata to implement the main body of SILCA.

To solve these problems we use the reinforced random search method described as follows. The value areas of x and y are represented by

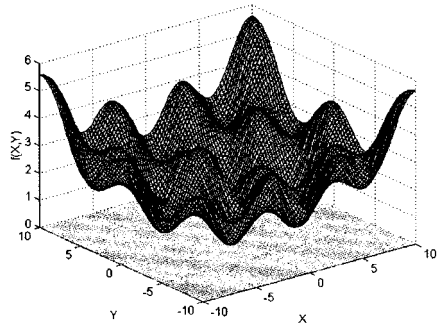
$$L=\left\{\underbrace{l_0^x,l_1^x,\ldots,l_{nx}^x}_{for\ x},\underbrace{l_0^y,l_1^y,\ldots,l_{ny}^y}_{for\ y}\right\} \quad \cdots\cdots\cdots (16)$$

$$l_i^j\in\{0,1\}, i=0,1,\ldots,nj, j=x,y$$

Where, the l_i^j is depending on the encoding method. We use the gray code encoding method to encode the x and y to the vector L. Therefore, $GRAY^{-1}(l_0^x,l_1^x,\ldots,l_{nx}^x)$, $GRAY^{-1}(l_0^y,l_1^y,\ldots,l_{ny}^y) \in [-10,10]$,$GRAY^{-1}(l)$ is an gray decoding function that decodes the vector L to a scalar value.



(a)TASK 1



(b)TASK 2

Fig. 5. The shapes of the two test functions

The variable structure of hierarchical learning automata (VSHLA) with three levels is shown as Fig. 6.

On the first level there is one stochastic automaton with two actions $y \in \{y^1, y^2\}$. When the automata on the first level selects the action $y^i(i = 1, 2)$ at the time n, i.e. $y(n) = y^i$, the ith automaton on the second level will be fired corresponding to the selection of the action. The automata on the second level have four actions, which will fire the automata on the third level. For the automata on the third level, the action corresponds to the subset of a gray encoded vectors of x and

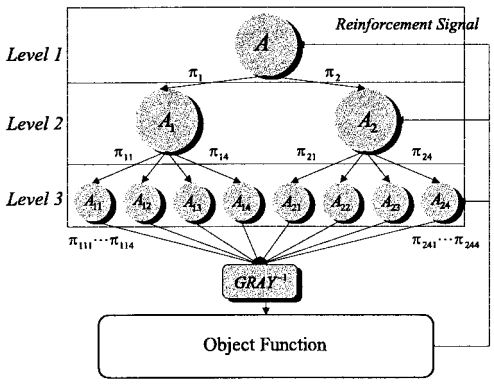


Fig. 6. The variable structure of hierarchical learning automata for function optimization problems.

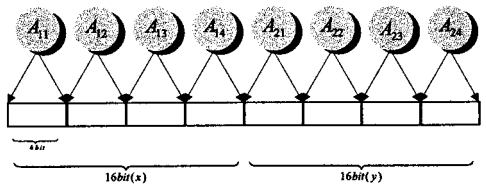


Fig. 7. Reinforced random search area for each automaton.

Table 1. The meaning of each action

Level		Actions			
1	y	y_1	y_2	y_3	y_4
	update	-	A_1	A_2	A_1, A_2
2	y	y_1^i	y_2^i	....	y_{16}^i
	update	-	A_1^i	....	$A_1^i \cdots A_4^i$
3	y	y_1^{ij}	y_1^{ij}	....	y_1^{ij}
	update	bit 0	bit 1	....	bit 0 ... bit4

The symbol "-" in Table 1 means don't fire or change any automata or bits in lower levels.

y (Fig.7).

First level:

$A = \{y, r, \phi, p, o, T\}$, where $y \in \{y_1, \ldots, y_4\}$ is the action (output), r is the input, ϕ is the state set, $p = \{p_1, \cdots, p_4\}$ is the probability distribution on the action set.

Second level:

$A_i = \{y^i, r, \phi^i, p^i, o^i, T^i\}, i = 1, 2$, where $y^i \in \{y_1^i, \ldots, y_{16}^i\}$, $p^i = \{p_1^i, \cdots, p_{16}^i\}$

Third level:

$A_{ij} = \{y^{ij}, r, \phi^{ij}, p^{ij}, o^{ij}, T^{ij}\}, i = 1, 2; j = 0, \cdots 3$, where $y^{ij} \in \{y_1^{ij}, \ldots, y_{16}^{ij}\}$, $p^{ij} = \{p_1^{ij}, \cdots, p_{16}^{ij}\}$

In our simulations we encode the x and y as 16-bit 0-1 vectors, i.e. in the eq. (16) $nx = 15$,$ny = 15$, and split the each vector into four pieces (Fig.7). Therefore the random search is implemented as Table 1.

Each bit on the L is turn around 0 and 1 with the probability:

$$m_{ijk} = \lambda \cdot \pi_i \cdot \pi_{ij} \cdot \pi_{ijk} \quad \cdots\cdots\cdots\cdots\cdots\cdots (17)$$

$$i = 1, 2\ ,\ j = 1, 2, 3, 4\ ,\ k = 1, 2, 3, 4$$

where $0 < \lambda < 1$ is a learning parameter. From Table 1, we can calculate the π_i,π_{ij},π_{ijk} as follows.

Let

$$\pi' = [\pi_1\ \ \pi_2]^T \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (18)$$

$$\pi'' = \begin{bmatrix} \pi_{11} \cdots \pi_{14} \\ \pi_{21} \cdots \pi_{24} \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (19)$$

$$\pi''' = \begin{bmatrix} \pi_{111} & \cdots & \pi_{114} \\ \vdots & \vdots & \vdots \\ \pi_{141} & \cdots & \pi_{144} \\ \pi_{211} & \cdots & \pi_{214} \\ \vdots & \vdots & \vdots \\ \pi_{241} & \cdots & \pi_{244} \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (20)$$

$$P' = \begin{bmatrix} p_2 + p_4 & 0 \\ 0 & p_3 + p_4 \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (21)$$

$$P'' = \begin{bmatrix} p_1^1 & \cdots & p_{16}^1 \\ p_1^2 & \cdots & p_{16}^2 \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (22)$$

$$P''' = \begin{bmatrix} p_1^{10} & \cdots & p_{16}^{10} \\ \vdots & \cdots & \vdots \\ p_1^{13} & \cdots & p_{16}^{13} \\ p_1^{20} & \cdots & p_{16}^{20} \\ \vdots & \cdots & \vdots \\ p_1^{23} & \cdots & p_{16}^{23} \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots (23)$$

$$U = \begin{bmatrix} 0101010101010101 \\ 0011001100110011 \\ 0000111100001111 \\ 0000000011111111 \end{bmatrix} \quad \cdots\cdots\cdots\cdots\cdots\cdots (24)$$

then,

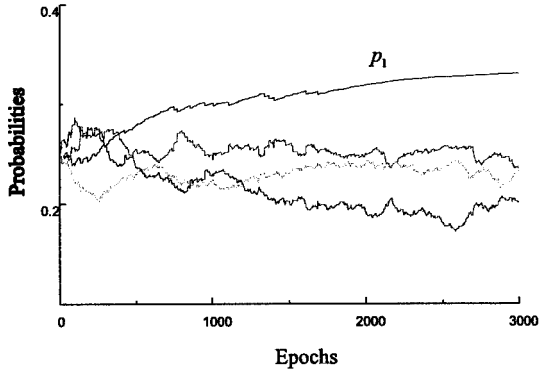
$$\pi' = P'\ ,\ \pi'' = P''U^T\ ,\ \pi''' = P'''U^T \quad \cdots\cdots (25)$$

For each automaton we use the linear reward-penalty scheme [2] [3] to update the action probabilities.

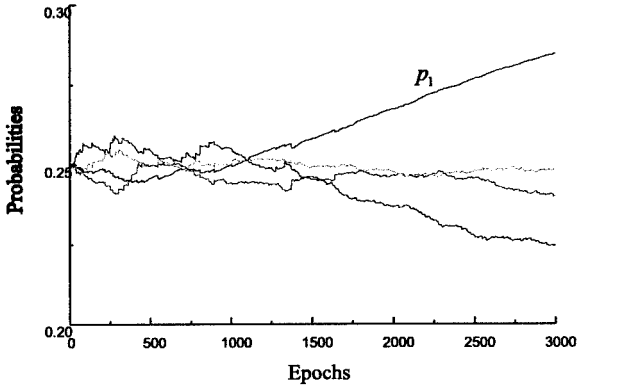
Our simulations start with one automaton that includes a universal constructor with two out-going cells and a VSHLA. We set the searching area $[a, b] \in [-10, 10]$ randomly. Once the state transition on the constructing loop has visited an out-going cell, we can get the following two processes.

- *Sequential process*: if the local searching on the parent is finished, and a local minimum has been found, then, the parent automaton begins to construct two offspring on the other areas and plays the same things on the offspring;
- *Partially parallel process*: the parent will construct an offspring and carry out the local searching independently.

Here, the simulation results for partially parallel process method are shown in Figs. 8, 9 and 10. The simulation starts on the area $[-7, -5]$ with one SILCA. The size of the inner sheath loop is set to 1/20 simulation times for one offspring. After 40 and 32 generations, the whole local minima have been found for TASK 1 and TASK 2 respectively. Fig.8 shows the evolution of the stochastic automaton's probabilities in level 1. Fig.9 shows the evolution of the probabilities of the offspring that near the global minimum area. Fig.10(a) shows the convergence corresponding to various learning parameters, and Fig.10(b) shows the local minima found by each offspring. The results show that using the SILCA we can search the full solution space to find the optimum.

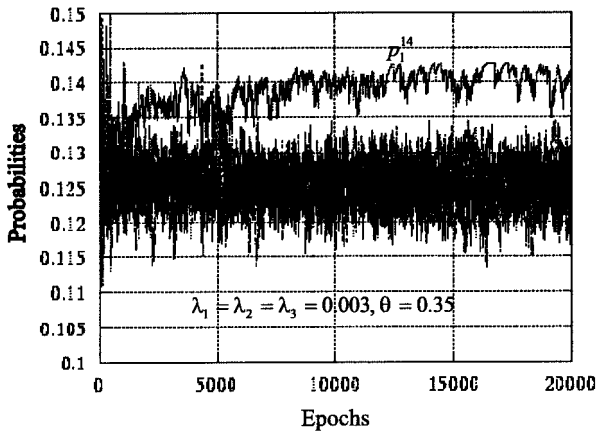


(a) TASK 1

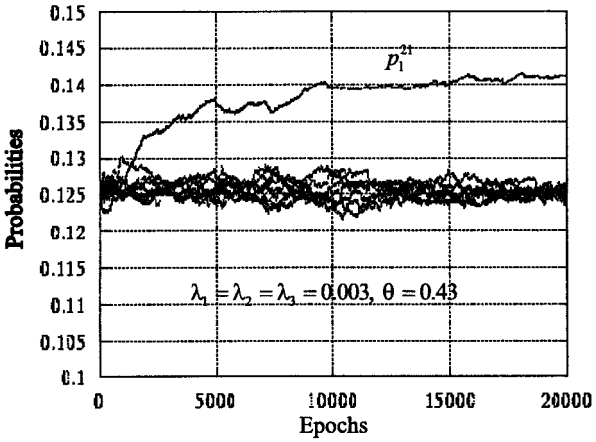


(b) TASK 2

Fig. 8. Evolution of the stochastic automaton's probabilities in level 1.

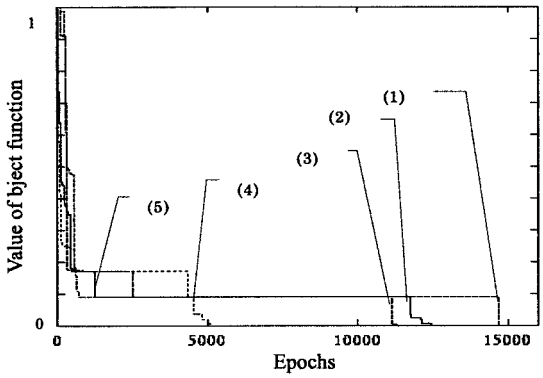


(a) TASK 1

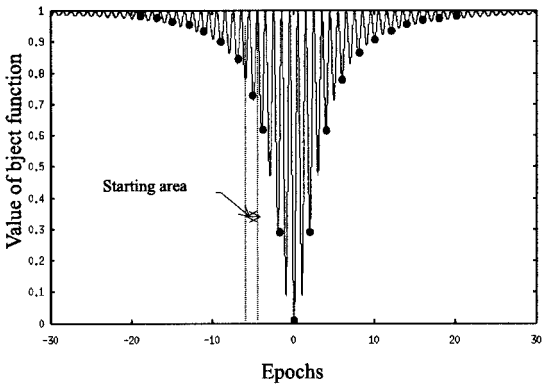


(b) TASK 2

Fig. 9. Evolution of the stochastic automaton's probabilities in level 3.



(a)Convergence corresponding to various learning parameter (TASK1). (1) $\lambda_1 = \lambda_2 = \lambda_3 = 0.001, \theta = 1$ (2) $\lambda_1 = \lambda_2 = \lambda_3 = 0.003, \theta = 1$ (3) $\lambda_1 = \lambda_2 = \lambda_3 = 0.003, \theta = 0.6$ (4) $\lambda_1 = 0.001, \lambda_2 = 0.002\lambda_3 = 0.001, \theta = 0.45$ (5) $\lambda_1 = 0.001\lambda_2 = 0.015\lambda_3 = 0.002, \theta = 0.53$



(b) The local minima found by each offspring (TASK1)

Fig. 10. Simulation results.

6. Conclusions

We have presented the self-improving learning cellular automata (SILCA), and we have shown that SILCA has the emergent property for function optimization. The SILCA consisting two parts: the main body that are implemented by the variable structure of hierarchical learning automata using the reinforced random search method, and the universal constructor implemented by the simplified Langton's periodic emitter.

Because using the SILCA we can search the solution space carefully, and each offspring can carry out the local searching parallel, we can use the parallel computer to implement the SILCA and solve the practical func-

tion optimization problems quickly. The results of computer simulations show the effectiveness of the SILCA.

(Manuscript received June 2, 2000)

References

(1) J. von Neumann: The Theory of Self-Reproducing Automata, *A. W. Burks, ed, University of Illinois Press, Urbana*(1966).

(2) K.S. Narendera & M.A.L. Thathachar: Learning Automata - An Introduction, *Prentice-Hall International* (1989).

(3) K.S. Narendera & M.A.L. Thathachar: Learning Automata - A survey, *IEEE Trans. on SMC*, Vol.4, No.4, pp.323-334 (1974).

(4) R.J.Willianms: Toward a Theory of Reinforcement-Learning Connectionist Systems, *Technical Report NU-CCS-88-3, Northeastern University* (1988).

(5) A.G.Barto: Learning by Statistical Cooperation of Self-interested Neuron-like Computing Elements, *Human Neurobiology*, Vol.4, pp.229-256 (1985).

(6) F.Qian and H.Hirata: Stochastic Learning Cellular Automata, *Proc. of the 26th ISCIE International Symposium*, pp. 107-112,(1994).

(7) F.Qian and H.Hirata: A Parallel Learning Automata for Combinatorial Optimization Problems, *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pp.553-558 (1996)

(8) E. F. Codd: Cellular Automata, *Academic Press, New York* (1968).

(9) C.G. Langton: Self-Reproduction in Cellular Automata, *Physics*, Vol. 10D, pp. 135-144 (1984).

Fei Qian (Member) Fei Qian recived a Master's degree from Tokyo Institute of Technology, Japan in 1987 and Ph.D. in Systems Engineering from Chiba University, Japan in 1991. From April 1991 to April 1995, he served as a Lecturer in the Department of Electronics Engineering and then a Associate Professor in the Department of Computer Science, Hiroshima Denki Institute of Technology. He is currently a Professor of Department of Computer Science at Hiroshima Kokusai Gakuin University. His current research interests include modeling and analyses of large-scale systems, and fundamental theory of learning automata, parallel distributed systems, such as computer network computing systems and neurocomputers. He is a member of IEEE, SICE, IEICE.

Yue Zhao (Member) Yue Zhao received the M.S. degree in 1985, and the Ph.D degree in 1988, all from the Tokyo Institute of Technology. Since 1993 he has been employed as a Lecturer in the Department of Computer Science at Hiroshima Kokusai Gakuin University. His research interests include combinatorial optimization, and parallel computing. Dr. Zhao is a member of IEICE, IPSJ, JSAI, JSCES.

Hironori Hirata (Member) Hironori Hirata received a Master and a Doctor in Engineering from Tokyo Institute Technology, Japan in 1973,1976 respectively . He is a professor of the Artificial Science Division, Graduate School of Science and Technology, Chiba University, Japan. His current research interests include modeling, analysis, and synthesis of large-scale systems, especially ecological systems and emergent systems. He is also interested in optimization and learning. He is a member of IEEE, ISEM, INNS, IEICE, SICE, ISCIE, IPSI.