Learning not to Fail
– Instance-Based Learning from Negative Reinforcement*

Tatsuo Unemi
Laboratory for International Fuzzy Engineering Research
Siber Hegner Bldg., 1-89 Yamashita-cho, Naka-ku, Yokohama, 231 JAPAN
Telephone +81 45 212 8231
FAX +81 45 212 8255
E-mail unemi@fuzzy.or.jp

Department of Information Systems Science, Soka University
236-1 Tangi-cho, Hachioji-shi, Tokyo 192 JAPAN
Telephone +81 426 91 9429
FAX +81 426 91 9312

1992

Abstract

We propose an instance-based learning algorithm named IBRL3 which acquires some kind of control rules not to fail in a dynamic environment, and we examine its performance via application to the cart-pole balancing problem. In this algorithm, a tuple of input, output and preference value of each execution cycle are stored in memory verbatim, and the action of each cycle is decided by retrieving the nearest neighbor of the current input data. The number of stored instances is reduced by replacing the nearest but less reliable instance by new one. Experimental results of computer simulation show that IBRL3 is robust for distinct settings of parameters and for noisy environments, and is efficient enough to apply to some kinds of real-time control problems.

Keywords: Applied adaptive behavior, Learning control, Cart-pole balancing problem, Instance-based learning, Reinforcement learning method.

1 Introduction

We propose an instance-based learning algorithm named IBRL3 which can be applied to control some kinds of unstable systems, and evaluate its performance by applying it to the cart-pole balancing problem.

On the contrast of inductive learning, in an instance-based learning, the learner makes no modification for its inputs and outputs to store them into the memory. It has neither rule nor equation to model its environment, and do no generalization. Instead, it retrieves the instances of high similarity with current input to decide the output. This sort of approach have been examined since the dawn of AI research[1], and many fields of application have been challenged, such as transforming English words to phonetic signs[2], machine translation[3], diagnostic system[4], simple robot control[5, 6], supervised classification problems[7] and so on, though they were under many kinds of different terms, such as rote learning, memory-based reasoning, instance-based learning, and so on. A recent literature by Salzberg[8] positioned instance-based learning as one kind of exemplar-based learning.

We have studied learning mechanisms from a standing point of simulating adaptive behavior of organisms, and have proposed instance-based learning mechanisms named IBRL1 and IBRL2[9] which can be applied to control problems that is the domain for reinforcement learning. It was evaluateded by applying an artificial insect surviving in a two dimensional Euclidean world simulated on the computer. In this paper, we propose an extended version of the algorithm which can learn to control the unstable inverted pendulum.

The cart-pole balancing problem is one of the popular testbeds for reinforcement learning methods, as it has been mentioned by many researchers, such as Michie and Chambers[10], Barto, Sutton, and Anderson[11], Selfridge, Sutton and Barto[12], Anderson[13], and so on. Recently, from the side of genetic algorithms, some researchers are challenging this problem, such as Odetayo and McGregor[14], Whitley, Dominic and Rajarshi[15] and Koike *et al*[16]. An instance-based approach was tried by Connel and Utgoff in their learning system named CART[17]. Sammut stated in his paper with experimental comparison of some of these approaches that CART provides the best learning performance on this problem domain, though it includes a number of domain specific heuristic strategies and parameters, and needs much amount of CPU time[18]. IBRL3 includes less domain specific heuristics and parameters and needs less CPU time than CART, spending a little victim of the learning performance; by employing a different strategy but a little similar to that of BOXES by Michie[10].

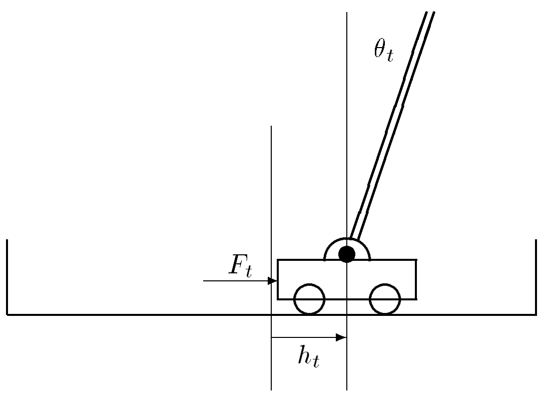

Figure 1: The cart-pole system.

We will describe the specification of the learning task, a detail of the learning algorithm, experimental results, and then comparison with other algorithms in the following sections.

2 The Cart-Pole Balancing Problem

The cart-pole balancing problem, namely the inverted pendulum problem, have often been mentioned as a typical control problem. Here we employ the problem specification used by Anderson[13] as the learning task. Figure 1 shows a schematic illustration of the cart-pole system.

State of the cart-pole system at time t is represented by the following four numerical parameters:

h_t	the position of the cart on the track,
$\dot{h}_t$	the velocity of the cart,
θ_t	the angular position of the pole, and
$\dot{\theta}_t$	the angular velocity of the pole.

The learner receives the vector of these four values as the input from the environments at each time step. The output from the learner, that is the control variable, is the force F_t which will be applied to the cart, of which value is either LEFT or RIGHT of fixed forces. It has no continuous numerical value. (See appendix for detail about the computer simulation.) In addition to the above four values, the reinforcement R_t is given to the learner, of which value becomes -1 when the cart reaches the end of track or the pole has fallen down, otherwise 0. The objective of learning is to get less frequently to suffer $R_t = -1$.

The learning is done through repeated trials for controlling the cart-pole system. The procedure of one trial is as follows.

1. Start with the cart at the center of track and with the pole standing vertically.
2. Control the cart-pole system.
3. Stop when the cart reaches the end of track or when the pole has fallen down.

If the learning is effective, the number of steps in one trial increases while iterating trials.

3 Learning Algorithm

The basic idea of instance-based learning can be said as follows.

- Store the tuples of input and output data in memory verbatim over the experiences.
- Select the best output by retrieving the instances from memory which is similar to the current situation.

In supervised classification problems, the output can be the category name of the most similar instance retrieved, the nearest neighbor. However, in control problems with delayed reinforcement, it becomes somewhat difficult to select the best action because the utility of the decision at each step is given later and is not always reliable.

We call the learning algorithm proposed here IBRL3 as the successor of IBRL1 and IBRL2 previously developed. In IBRL3, like BOXES, it makes no modification in memory until the trial finishes, and after that the instances given during the trial are stored in memory, because the instances on the current trial are given no evaluation and they are useless for the decision procedure.

The main program of IBRL3 can be drawn as follows.

Algorithm 1

```
program **IBRL3**;
repeat begin
    $t := 0$;
    repeat begin
        $t := t + 1$;
        $x :=$ **GetSenseData**;
        $y :=$ **Policy**($x$);
        **TakeAction**($y$);
    end until **GetReinforcement**() $= -1$
    **ModifyMem**
end forever.
```

The input and output data at each step is recorded in queue, Q, in function **Policy**, and they are moved into the memory in procedure **ModifyMem** after the reinforcement becomes -1.

IBRL3 uses a number of sets as memory each of which is corresponding to a distinct alternative for output, LEFT and RIGHT on the cart-pole balancing problem. We denote the set corresponding to the output data y by M_y. An element in this set is a tuple of input data and its preference value P_i, $((h_i, \dot{h}_i, \theta_i, \dot{\theta}_i), P_i)$. Here we call this element *instance*. In the cart-pole balancing problem, P_i can be the expected number of steps till falling down. Considering general cases for n dimensional input data, we denote an instance by

$$I_i = ((u_{i1}, u_{i2}, \ldots, u_{in}), P_i).$$

On output data, we denote the set of alternatives by $\mathcal{O}$ so as to make it possible to mention more than two candidates. In the case of the cart-pole balancing problem, $\mathcal{O} = \{\text{LEFT}, \text{RIGHT}\}$.

We describe a detail of the decision procedure **Policy** and the memorizing procedure **ModifyMem** in the rest of this section.

3.1 Decision making

The decision procedure named **Policy** is the central mechanism in the performance module. In each step, this procedure retrieves the instances involving the nearest input data of the current input x from each of memory M_y, and selects the best one in terms of the preference value P_i as the output of next step. When M_y is empty or the distance to the nearest instance is over the threshold, it assumes the preference value to be an estimated average of previous experiences. The final phase of the procedure is to record the current input, the selected output, the nearest instance with the highest value and that distance into a queue as its tth element Q_t. The detail is as follows.

Algorithm 2

```
function **Policy**($x$)
for all $v$ in $\mathcal{O}$ do begin
    find $I_i$ in $M_v$ which minimize $\mathcal{D}(I_i, x)$;
    if $M_v = \emptyset$ or $\mathcal{D}(I_i, x) \geq 1$
        then begin $p_v := E;\ N_v := \emptyset;\ D_v := \infty$ end
        else begin $p_v := P_i;\ N_v := I_i;\ D_v := \mathcal{D}(I_i, x)$ end
end
find $y$ in $\mathcal{O}$ which maximize $p_y$,
    where if more than one candidates for $y$ exist
    then randomly select one among them;
$Q_t := (x, y, N_y, D_y)$;
return $y$.
```

$\mathcal{D}(I_i, x)$ is a normalized distance between an instance I_i and an input data x which defined by the equation:

$$\mathcal{D}(I_i, x) \ = \ \frac{1}{2n} \sum_{j=1}^{n} \frac{(u_{ij} - x_j)^2}{s_j^2}$$

where s_j^2 is the sample variance over the values u_{ij} of all instances in current memory. Sample variance is defined as

$$s_j^2 \ = \ \frac{1}{N-1} \sum_{k}^{N} (\bar{u}_j - u_{kj})^2$$

where N is the number of samples. As the following equations indicate, the average value of $\mathcal{D}(I_i, I_k)$ becomes 1 in any type of distribution.

$$\begin{aligned} \mathrm{E}_{\mathcal{D}} &= \frac{1}{2n} \cdot \frac{2}{N\,(N-1)} \cdot \sum_{i=2}^{N} \sum_{k=1}^{i-1} \sum_{j=1}^{n} \frac{(u_{ij} - u_{kj})^2}{s_j^2} \\ &= \frac{1}{2n} \cdot \frac{2}{N\,(N-1)} \cdot Nn\,(N-1) \\ &= 1. \end{aligned}$$

The threshold value 1 for the distance have come from this fact. It is of course possible to employ another sort of measure for distance, but the square of Euclidean distance takes relatively less computational time and normalizing by sample variance has an advantage described above.

E is a global variable of which value is the estimated average of preference. In each end of trials, it is renewed by

$$E \ := \ \alpha \cdot E + (1 - \alpha) \cdot \beta \cdot T$$

where T is the number of steps of the trial, and α and β are constants of [0, 1]. We use $\alpha = 0.8$ and $\beta = 0.5$ in our experiments described later.

3.2 Memorizing

After one trial stopped, the learner modifies its memory referring to the history of the trial recorded in the queue. The basic strategies of this operation are:

- to memorize the instances so as to cover the state space with an appropriate density, and
- to manage the preference value of each instance so as to estimate the number steps till a negative reinforcement would be given if it selected the output value associated with the instance.

These strategies can be implemented in the following procedure.

- If the distance is short and the new data is better than the known instance, replace the known instance by new data.
- If the distance is short but the new data is worse than the known instance, reduce the preference value of the known instance.
- If there is no instance near enough, add the new data into memory.

The detail algorithm is as follows.

Algorithm 3

```
procedure **ModifyMem**;
for $k$ in $[1..t]$ do begin
    $(x, y, I_i, d) := Q_k$;
    if $d < \rho$ then
        if $t - k > P_i$
            then replace $I_i$ in $M_y$ by $(x, t - k)$
            else reduce $P_i$
    else add $(x, t - k)$ into $M_y$
end.
```

For all elements in the queue Q_k $(k = 1, 2, \ldots, t)$, if the distance d, the fourth element of Q_k, is shorter than the threshold ρ, and the preference value P_i of the instance I_i, the third element of Q_k, is less than $t - k$, then replace I_i in the set M_y by a new instance including the input data x, the first element of Q_k, and $t - k$ as its preference value, where the output datum y is the second element of Q_k. If P_i is greater than $t - k$ then reduce P_i. If the distance d is longer than ρ, add the new instance into M_y.

Reduction of the preference value is done by computing the weighted average among the current value and $t - k$ for all of k where Q_k involves I_i, where the weight of the current value is 1 and the weight of $t - k$ is $1 - \sqrt{d}$. The more similar, the greater weighted. If this type of reduction were omitted, the performance module could not avoid falling down from a cliff while walking along the course slitely different from the good route experienced before.

4 Experimental Results

Our experiments are under quite same settings as that done by Sammut[18], that is, the initial state of each trial is given as $|h_0| < 0.1m$, $|\theta_0| < 6°$ of uniformly distributed random numbers and both $\dot{h}_0$ and $\dot{\theta}_0$ are 0. And the negative reinforcement $R_t = -1$ is given when $|h_t| > 2.4m$ or $|\theta_t| > 12°$. We tried the following three cases of distinct values of the force applied to the cart.

Case1: $F_t = 10N$ when the output datum $y = \text{RIGHT}$, $F_t = -10N$ when $y = \text{LEFT}$.

Case2: $F_t = 5N$ when $y = \text{RIGHT}$, $F_t = -10N$ when $y = \text{LEFT}$.

Case3: Same settings as Case1 except noise e_t given by uniformly distributed random numbers of $|e_t| < 2$ is added, that is, $F_t = (\pm 10 + e_t)N$.

Case3, not mentioned by Sammut, is useful to expect the performance on real machines because there are always many factors not considered in the ideal simulation.

One *run* stops when the pole does not fall down for more than 10,000 steps while iterating trials. The result of run is the number of trials it includes. We evaluated the learning performance by the averaged number of trials over 100 runs on distinct random number sequences. For each of Case1, Case2 and Case3, the evaluation was done for distinct values of ρ, the threshold used in **ModifyMem**, from 0.2 to 1.0 increasing by 0.1. Table 1 shows the results which consists of averages, standard deviations, minimum values and maximum values of the number of trials and instances in memory. Figure 2 summarizes the averaged performances and the memory costs of IBRL3 on a variety of ρs.

The number of trials increases in the order of Case1, Case2 and Case3. In those cases, the number becomes minimum at around $\rho = 0.3$ or 0.4. This indicates that the performance gets worse in both cases of too small and too large values of ρ. The number of instances in memory which indicates the computational cost of time and space varies inversely as ρ. This result is natural because the chance of adding new data into memory must increase for less value of ρ. It is important that the results of Case2 and Case3 suggests robustness of IBRL3 for other settings and for noisy environments, which implies usefulness for real world applications.

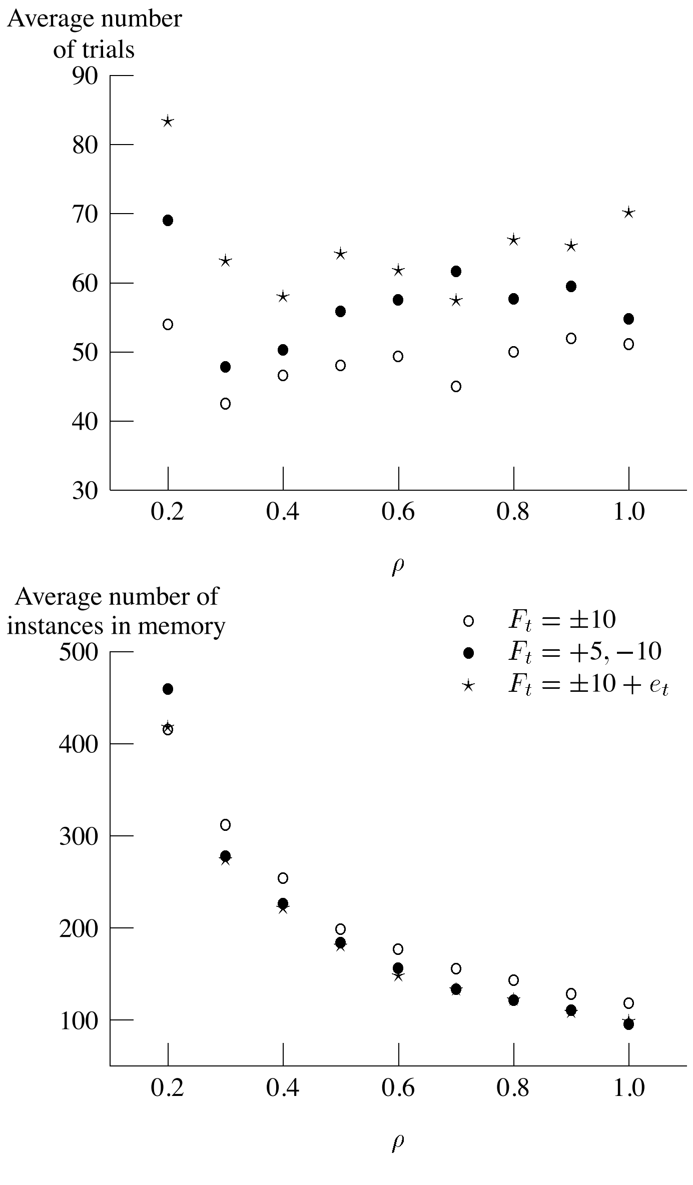

Figure 2: Average number of trials and instances in memory over 100 runs varying the threshold value ρ.

5 Comparison with Other Methods

Sammut compared BOXES by Michie[10], AHC[11, 12] by Barto and Sutton, and CART by Connel[17], in his experiments[18]. Table 2 shows comparison with that results on learning rate. As this table indicates, in Case1 CART shows the best performance and IBRL3 is the second, but in Case2 IBRL3 is the best. Sammut did not show the exact value for CART in Case2, but he pointed out that its took 58 steps in less severe condition. In BOXES, AHC and CART, the trials in Case2 are 3.7, 28 and 4.5 times of in Case1 respectively, but less than 1.2

times in IBRL3. This indicates that IBRL3 is more robust than the others against somewhat complex state space.

On the point of computational cost, IBRL3 takes proportional CPU time to the number of instances in memory in each step. But it can be reduced by increasing the value of ρ spending a little victim of the learning performance. In the experimental results, the number of instances is 87.22 in average and 624 in max for Case1 of $\rho = 0.3$ in which it shows the best performance. In the case of 376 instances, it took 3.03 milliseconds of CPU time par one step on SPARC Station IPC with 25MHz SPARC IU and FPU, which is short enough to control the real system. Note that the time step for computer simulation is 20 milliseconds. According to the paper by Sammut, CART takes too long CPU time to control for the real system.

6 Conclusion

We proposed an instance-based learning algorithm named IBRL3 which can be applied to learning control for the cart-pole system, and confirmed its performance through the experiments on computer simulation. Experimental results shows that IBRL3 is robust for distinct settings of parameters and for noisy environments, and is efficient enough for real machine applications.

This approach is generally applicable to the learning task with delayed negative reinforcement, where the input is represented in a vector of numerical values and the output is a symbol selected from *a priori* known set. When we consider to apply this algorithm to more complex areas, such as a large number of factors in the input or a large number of alternatives for output, both learning rate and computational cost would get worse. But fortunately, more efficient algorithms have been proposed to find the nearest neighbor in n-dimensional Euclidean space as used in Moore's system[6]; though there may be no settlement for learning rate.

One additional condition for the applicable task is that the optimal or pseudo-optimal action must be determined from the current input at each step independently from the context. If the learning task requires context sensitive decision making, the retrieval of known instances must mention the context. This problem will be tackled in our next work.

Acknowledgement

The author thanks Akihiko Koike, Hiroaki Ogura, Yasuhiro Uekou and Hideaki Shono at Nagaoka University of Technology for their help to survey the related fields, and appreciates the participants of 14th Intelligent Systems Symposium held at Okayama University in Japan for their helpful discussion. This research was supported by AI Research Foundation in Japan grant No. 4 in 1991.

References

[1] Samuel, A. L.: Some Studies in Machine Learning Using the Game of Checkers, *IBM Journal on Research and Development*, Vol. 3, pp. 210–229, (1959).

[2] Stanfill, C. and D. Waltz: Toward Memory-Based Reasoning, *Communications of the ACM*, Vol. 29, pp. 1213–1228, (1986).

[3] Satoh, S.: MBT2: A Method for Combining Fragments of Examples in Example-Based Translation, *Journal of Japanese Society for Artificial Intelligence*, Vol. 6, pp. 861–871, (1991) in Japanese.

[4] Waltz, D. L.: Applications of the Connection Machine, *IEEE Computer*, Vol. 20, pp. 85–97, (1987).

[5] Mason, M. T., A. D. Christiansen and T. M. Mitchell: Experiments in Robot Learning, *Proceedings of the Sixth International Workshop on Machine Learning*, pp. 141–150, (1989).

[6] Moore, A. W.: Acquisition of Dynamic Control Knowledge for a Robotic Manipulator, *Proceedings of the Seventh International Conference on Machine Learning*, pp. 244–252, (1990).

[7] Aha, D. W., D. Kibler and M. K. Albert: Instance-Based Learning Algorithm, *Machine Learning*, Vol. 6, pp. 37–66, (1991).

[8] Salzberg, S.: A Nearest Hyperrectangle Learning Method, *Machine Learning*, Vol. 6, pp. 251–276, (1991).

[9] Unemi, T.: Instance-based Reinforcement Learning Method, *Journal of Japanese Society for Artificial Intelligence*, (to appear.) in Japanese.

[10] Michie, D. and R. A. Chambers: Boxes: An Experiment in Adaptive Control, in E. Dale and D. Michie (eds), *Machine Intelligence 2*, Oliver & Boyd, Edinburgh, pp. 137–152, (1968).

[11] Barto, A. G., R. S. Sutton and C. W. Anderson: Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 13, No. 5, pp. 834–846, (1983).

[12] Selfridge, O. G., R. S. Sutton and A. G. Barto: Training and Tracking in Robotics, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 670–672, (1985).

[13] Anderson, C. W.: Strategy Learning with Multilayer Connectionist Representations, *Proceedings of the Fourth International Workshop on Machine Learning*, pp. 103–114, (1987).

[14] Odetayo, M. O. and D. R. McGregor: Genetic Algorithm for Inducing Control Rules for a Dynamic System, *Proceedings of the Third International Conference on Genetic Algorithm*, pp. 177–182, (1989).

[15] Whitley, D., S. Dominic and D. Rajarshi: Genetic Reinforcement Learning with Multilayer Neural Networks, *Proceedings of the Fourth International Conference on Genetic Algorithm*, pp. 562–569, (1991).

[16] Koike, A, H, Ogura, T. Unemi and Y. Yoshitani: Control for an Unstable System using a Genetic Algorithm, *Proceedings of the 14th Intelligent Systems Symposium*, Society of Instruments and Control Engineering in Japan, pp. 169–175, (1991) in Japanese.

[17] Connell, M. E. and P. E. Utgoff: Learning to Control a Dynamic Physical System, *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 456–460, (1987).

[18] Sammut, C.: Experimental Results from an Evaluation of Algorithms that Learn to Control Dynamic Systems, *Proceedings of the Fifth International Conference on Machine Learning*, pp. 437–443, (1988).

Appendix: Simulation of the Cart-Pole System

The cart-pole system is simulated according to the following equations of motion[13]:

$$\ddot{\theta}_t = \frac{g \sin\theta_t + \cos\theta_t \left[\frac{-F_t - m_p l \dot{\theta}_t^2 \sin\theta_t}{m_c + m_p}\right]}{l\left[\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p}\right]}$$

$$\ddot{h}_t = \frac{F_t + m_p l \left[\dot{\theta}_t^{\,2} \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right]}{m_c + m_p}$$

where

$$\begin{aligned} g &= 9.8 m/s^2 = \text{acceleration due to gravity,} \\ m_c &= 1.0kg = \text{mass of the cart,} \\ m_p &= 0.1kg = \text{mass of the pole,} \\ l &= 0.5m = \text{distance from center of mass of pole to the pivot.} \end{aligned}$$

The simulater computes numerically approximated values of state variables at each time step sliced by $\tau = 0.02$ seconds using Euler's method with the following state equations:

$$\begin{aligned} h_{t+1} &= h_t + \tau \dot{h}_t, \quad \dot{h}_{t+1} = \dot{h}_t + \tau \ddot{h}_t, \\ \theta_{t+1} &= \theta_t + \tau \dot{\theta}_t, \quad \dot{\theta}_{t+1} = \dot{\theta}_t + \tau \ddot{\theta}_t. \end{aligned}$$

The time length to apply force F_t applied to the cart is 0.02 seconds the same value of τ.

Table 1: Experimental results. Averages, standard deviations, minimum values and maximum values of the number of trials and instances in memory over 100 runs.

Case1: RIGHT$= 10N$, LEFT$= -10N$

	Trials				Instances			
ρ	Average	S.D.	Min	Max	Average	S.D.	Min	Max
0.2	53.88	52.42	7	390	414.56	146.85	81	788
0.3	42.43	26.54	6	211	311.17	87.22	79	624
0.4	46.54	33.67	10	198	253.22	124.13	124	1017
0.5	48.00	40.63	6	237	198.40	42.35	95	305
0.6	49.20	37.06	4	213	176.53	46.08	66	343
0.7	44.88	32.30	4	173	155.65	36.73	62	248
0.8	49.93	34.54	7	168	143.29	27.16	95	260
0.9	51.89	37.89	7	208	127.86	26.66	55	203
1.0	51.01	35.29	6	186	118.09	22.85	54	208

Case2: RIGHT$= 5N$, LEFT$= -10N$

	Trials				Instances			
ρ	Average	S.D.	Min	Max	Average	S.D.	Min	Max
0.2	68.97	53.94	13	288	458.58	177.24	162	1437
0.3	47.73	31.07	8	169	277.58	72.32	119	501
0.4	50.16	38.00	7	191	225.84	83.66	71	770
0.5	55.75	44.82	7	213	183.88	60.52	99	582
0.6	57.39	49.77	8	295	155.91	41.09	49	345
0.7	61.58	49.25	9	289	133.28	30.98	60	223
0.8	57.53	41.42	8	323	121.05	28.10	72	201
0.9	59.35	45.44	10	296	110.29	26.85	49	240
1.0	54.73	42.92	12	285	95.23	20.03	51	164

Case3: RIGHT$= (10 + e_t)N$, LEFT$= (-10 + e_t)N$, $(|e_t| < 2)$

	Trials				Instances			
ρ	Average	S.D.	Min	Max	Average	S.D.	Min	Max
0.2	83.38	69.32	6	431	418.12	109.75	84	710
0.3	63.17	56.83	10	266	274.68	82.67	105	560
0.4	58.03	53.72	13	326	221.84	100.52	114	1076
0.5	64.19	52.94	14	365	180.97	47.57	90	427
0.6	61.83	42.80	12	234	148.69	30.61	81	253
0.7	57.43	34.95	6	204	133.61	29.36	61	213
0.8	66.17	47.40	6	262	122.68	30.33	63	257
0.9	65.40	53.89	12	290	109.23	21.93	68	185
1.0	70.14	46.94	11	240	98.96	20.69	62	172

Table 2: Comparison with the experimental results by Sammut. Each value except IBRL3 is the averaged number of trials of five runs. The value of IBRL3 is of 100 runs where $\rho = 0.3$.

	BOXES	AHC	CART	IBRL3
Case1	225	90	13	42.43
Case2	837	2562	> 58	47.73