Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 9, No. 4, 2021, 487-496.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

Improving Speed and Efficiency of Dynamic Programming Methods through Chaos

Habib Khodadadi and Vali Derhami*

Computer Engineering Department, Yazd University, Yazd, Iran.

Article Info

Article History:
Received 04 February 2021
Revised 07 July 2021
Accepted 13 August 2021

DOI:10.22044/JADM.2021.10520.2191

Keywords:
Chaos, Dynamic Programming, Logistic Chaotic System, Policy Iteration, Reinforcement Learning, Value Iteration.

**Corresponding author:*
vderhami@yazd.ac.ir (V. Derhami).

Abstract

A prominent weakness of the dynamic programming methods is that they perform operations throughout the entire set of states in a Markov decision process in every updating phase. In this paper, we propose a novel chaos-based method in order to solve the problem. For this purpose, a chaotic system is first initialized, and the resultant numbers are mapped onto the environment states through initial processing. In each traverse of the policy iteration method, policy evaluation is performed only once, and only a few states are updated. These states are proposed by the chaos system. In this method, the policy evaluation and improvement cycle lasts until an optimal policy is formulated in the environment. The same procedure is performed in the value iteration method, and only the values of a few states proposed by the chaos are updated in each traverse, whereas the values of the other states are left unchanged. Unlike the conventional methods, an optimal solution can be obtained in the proposed method by only updating a limited number of states that are properly distributed all over the environment by chaos. The test results indicate the improved speed and efficiency of the chaotic dynamic programming methods in obtaining the optimal solution in different grid environments.

1. Introduction

Dynamic programming is a useful technique for solving problems. Within the framework of reinforcement learning and dynamic programming, the value iteration and policy iteration methods can be employed in order to solve some problems modeled on the Markov decision process (MDP). These methods try to iteratively update the values attributed to the states to approach the optimal value.

In the dynamic programming methods, a serious challenge is to perform the updating process, and operate on all sets of states. This problem is one of the major obstacles to the implementation of these methods because the time complexity of these types of algorithms is very high and their use is not cost-effective. In this case, the asynchronous dynamic programing languages can be used, for it is not necessary to traverse the entire state set. These algorithms support the values of states without considering any orders, and use the available values of the other states at the time of calculation. This class of algorithms provides a considerable flexibility in selecting the supported states [1].

In [2], the real-time dynamic programming (RTDP) has been proposed in order to overcome the problem of global search by evaluating only a subset of the state space. At each step of the trial (paths from the start state to a goal state), RTDP updates the current state, and changes the current state randomly according to the transition function. Each trial stops when a goal state is reached or a maximum number of steps is accomplished. The states unreachable from the start state are ignored in the trials, and therefore, are never updated. Other versions of this

algorithm have also been developed due to some of its flaws such as the lack of a convergence detection mechanism. For instance, the labeled-RTDP method has been developed in [3] in order to label the solved states and accelerate convergence by avoiding the unnecessary supporting procedures in those states. The bounded-RTDP (BRTDP) algorithm has been proposed in [4] in order to determine the upper and lower bounds of the optimal value function and use them to decide whether the current state value function reaches convergence so that further search could be performed in those areas of the state space with large difference. Another version of the BRTDP algorithm has been proposed in [5] in order to avoid the states with converged policies but no converged value functions in addition to identifying the states with converged policies. A novel version of the BRTDP algorithm has been employed in [6] in order to solve the dynamic resource routing problem by changing the upper and lower bounds adaptively.

The topological value iteration (TVI) algorithm has been developed in [7] in order to first divide every MDP into strongly connected components (SCCs) and then solve these components. In this method, all updates are done on the values of an SCC in every step, and the next SCC is selected when the previous one is converged. SCCs are converged in an inverted topological order. The topological policy iteration operations have been performed on the MDPs of limited parameters in [8]. Similar to [7], necessary updates are then applied in an inverted topological order after the strongly connected components are found.

The idea of prioritizing states based on the well-known Bellman error has been used in [9]. Here, the problem state space is divided into several clusters. In fact, the states of a partition are updated instead of applying global updates. A cluster is selected, and its state values are updated until convergence occurs. Only in the case of convergence, another cluster is selected. In every step, this method selects a partition with the highest dependency on the previously selected partitions. Another algorithm has been developed in [10] in order to use the referrals to one state from the other states as a criterion for prioritizing. The low priority states can be omitted here.

The heuristic search algorithms have also been employed in order to improve efficiency in solving the MDP problems. These algorithms try to use appropriate heuristic functions in order to find a solution faster and avoid unimportant updates. For instance, a heuristic method called LAO* has been introduced in [11] in order to solve an MDP problem that has two alternating stages. LAO* expands the best partial solution, and evaluates the states using the heuristic function. It then executes dynamic programming on the visited states to update their values and possibly revise the best current solution. Another heuristic algorithm called Anytime AO* has also been proposed in [12] to remove the flaws of the previous algorithms. It can achieve an optimal policy in the environment without having an admissible heuristic process.

In all the reviewed methods, the classic problems of dynamic programming may still exist while dealing with plenty of states and in large-scale problems.

In this work, chaos was employed in order to find the states that should be updated in the policy iteration or value iteration processes. The chaotic systems are known as effective methods in this problem due to having unique features such as sensitivity to the initial value, pseudo-randomness, unpredictability, non-periodic mechanism, and examination of different segments of the state space.

The research hypothesis is to employ chaos to improve the speed and efficiency of the classic methods of policy iteration and value iteration. This has never been utilized before, and can widely be applied in many techniques. Therefore, this manuscript focuses merely on the role of chaos in dynamic programming.

Section 2 gives a brief introduction of chaotic systems, and Section 3 discusses the reinforcement learning and dynamic programming briefly. The proposed method is presented in Section 4, and its calculative results are analyzed in Section 5.

2. Chaotic Systems

The chaos theory concerns the systems whose dynamics show such high sensitivity to changes in the initial values that it will be impossible to predict their future behavior.

The chaotic systems are non-linear systems that are very sensitive to their initial conditions, and show a pseudo-random behavior. Making a slight change in the initial conditions of such systems will lead to massive changes in the future; this phenomenon is known as the butterfly effect in the chaos theory. Despite their pseudo-random behavior, definability is an important feature of chaotic systems that have made then popular with many applications such as cryptography [13-16].

Many chaotic systems have been introduced so far. For instance, the Lorenz chaotic system [17] is based on the dynamic equations of real systems.

Another instance is the Chen chaotic system [18] that has no specific physical changes, and is merely a mathematical model.
Equation (1) shows the equations governing the Lorenz system.

$$\begin{cases} \dot{x} = a(y - x) \\ \dot{y} = bx - y - xz \\ \dot{z} = xy - cz \end{cases} \quad (1)$$

This system is chaotic if a = 10, b = 28, and c = 8/3.
The logistic system [19] is another chaotic system, in which the governing equations are shown as Equation (2):

$$x_{n+1} = \lambda x_n(1 - x_n) \quad (2)$$

Where x_0 is a value within the (1-0) range, from which the following values are obtained. Regarding the values of λ within [3.56-4], the system shows a chaotic behavior.
Figure 1 shows the behavior of the logistic system with the initial value of $x_0 = 0.52$ and $\lambda = 3.9999$; moreover, Table 1 indicates the first 25 numbers generated by the system. These numbers were approximated to four digits.
Evidently, the generated numbers were distributed properly in the space between 0 and 1, and the other segments of the space were visited after a few iterations. This is another feature of the chaotic systems. If the initial value presented in Figure 1 changes only to a small value, the following generated numbers will be very different from these numbers.
Figure 2 demonstrates the path on which a robot moves to find a target. Chaos was used in the equations through which the robot moves. Accordingly, the robot searches the space very well in order to find the target chaotically [20].

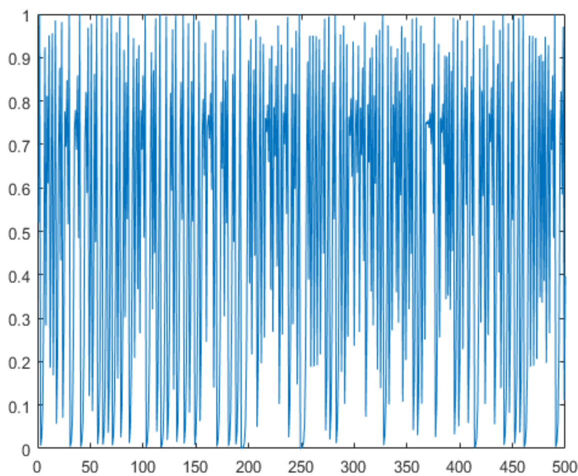

Figure 1. Chaotic behavior of the logistic signal in the first 500 iterations with $x_0 = 0.52$ and $\lambda = 3.9999$ (the horizontal dimension shows the number of iterations, whereas the vertical dimension indicates the resultant logistic signal in every iteration).

Based on the discussed features of chaos, the chaotic generated numbers can replace the random numbers. Despite benefiting from the pseudo-random behavior of this phenomenon, it is also possible to use its certainty. In other words, the same numbers can always be generated by giving every initial value.

Table 1. First 25 numbers of the logistic signal with $x_0 = 0.52$ and $\lambda = 3.9999$.

	1	2	3	4	5
Values 1 to 5	0.5200	0.9984	0.0065	0.258	0.1005
Values 6 to 10	0.3615	0.9233	0.2833	0.8121	0.6104
Values 11 to 15	0.9512	0.1857	0.6049	0.9560	0.1684
Values 16 to 20	0.5601	0.9855	0.0571	0.2153	0.6757
Values 21 to 25	0.8765	0.4329	0.9820	0.0709	0.2634

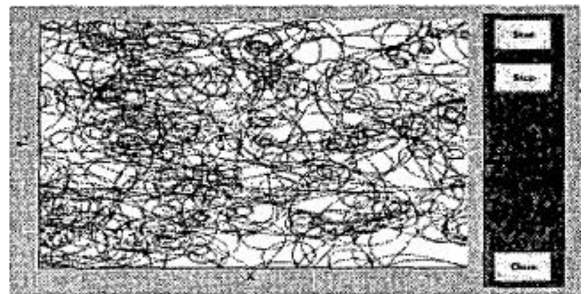

Figure 2. Robot's path towards the target through the Lorenz chaotic equation [20].

3. Reinforcement Learning and Dynamic Programming

Reinforcement learning means learning an appropriate action from a series of authorized actions for a particular situation based on the granted rewards or received penalties [1]. The key idea of reinforcement learning is to use the value functions to find appropriate policies. Dynamic programming is a method of reinforcement learning, in which the Bellman equation is employed in order to calculate the value of each state of an environment or the state-action value (Equations (3) and (4)). The values of other states are utilized to calculate the value of each state.

$$V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')] \quad (3)$$

$$Q^{\pi}(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')] \quad (4)$$

In these equations, Action a is selected from the state set s, and the next states of s' are the members of the state set. Moreover, $V^{\pi}(s)$ denotes the value of s under policy π, whereas $P^a_{ss'}$ and $R^a_{ss'}$ show the transmission probability and the expected value of the next reward, respectively. Furthermore, $\pi(s,a)$ refers to the

probability of selecting an action (a) in the state (s). Finally, γ is the discount factor.

The dynamic programming algorithms require a thorough model of the environment to formulate the optimal policies. Such a model is usually presented as an MDP.

Assume that there is an arbitrary policy (π) for a problem. The process of calculating the state value function (V^{π}) is called the policy valuation of π [1]. In this method, the initial approximation (V_0) is selected optionally, except for the values of the final states that should be selected zero if they exist. The next approximations are obtained from the Bellman equation as an updating rule for all states.

The process of formulating a new policy, which improves the value function of an initial policy from a relatively greedy perspective, is called the policy improvement method.

If the value of π is estimated, then V^{π} can be employed to improve π in case it is not optimal. Therefore, a better policy (π') is developed. It is then possible to calculate the value of $V^{\pi'}$ to improve the existing policy again and formulate a better policy (π"). Hence, a sequence of policies and value functions can be generated and improved through a normal procedure. This method of finding an optimal policy is called the value iteration, which is usually converged through multiple iterations. The algorithms 1, 2, and 3 present the pseudo-codes of policy evaluation, policy improvement, and policy iteration, respectively.

In the policy improvement pseudo-code, argmax_a denotes a value of a that maximizes the expression, i.e. selecting a greedy action. In fact, it selects an action that looks the best action in the short run (the next step) based on V^{π}.

A weakness of the policy iteration method is that it executes policy evaluation and policy improvement processes consecutively many times. If the optimal policy value function is estimated instead of the current policy, it is then possible to formulate the optimal policy after the optimal value function is obtained. This solution is called the value iteration method, in which the maximum value is selected from the resultant values of actions in each state. Algorithm 4 shows the pseudo-code of the value iteration method.

Algorithm 1: Pseudo-code of the iterative policy evaluation [1].

Function policyEvaluation (Inputs) **Return** Output
  Inputs: π//the policy to be evaluated
      V(s)//V(s)ϵR
  Repeat
    Δ = 0;
    For each sϵS
      v = V(s);
      $V(s) = \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$;
      Δ=max (Δ,|v-V(s)|);
    End for
  Until Δ<ε //ε is small positive threshold
  Output: V//V≈V^{π}

Algorithm 2: Pseudo-code of the policy improvement method [1].

Function policyImprovement (Inputs) **Return** Output
  Inputs: V(s)//V(s) ϵR
  For each sϵS
    $\pi(s) = \text{argmax}_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$
  Output: π(s)// π(s) ϵA(s): ∀sϵS

Algorithm 3: Pseudo-code of the policy iteration method [1].

Function policyIteration (Inputs) **Return** Output
  Inputs: π(s)// π(s) ϵA(s): ∀sϵS
      V(s)//V(s)ϵR
  Repeat
    $\pi' = \pi$
    V= policyEvaluation (π,V);
    π= policyImprovement (V);
  Until $(\pi = \pi')$
  Output: π(s)// π(s) ϵA(s): ∀sϵS

Algorithm 4: Pseudo-code of the value iteration method [1].

Function valueIteration (Inputs) **Return** Output
  Inputs: V(s)//V(s)ϵR
  Repeat
    Δ = 0;
    For each sϵS
      v=V(s);
      $V(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$;
      Δ = max (Δ,|v-V(s)|);
    End for
  Until Δ<ε //ε is small positive threshold
  $\pi(s) = \text{argmax}_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$
  Output: π(s)// π(s) ϵA(s): ∀sϵS

4. Proposed Method

The chaotic systems (*e.g.* logistic systems) are employed in order to prevent global updates in the dynamic programming algorithms. In the policy iteration method, a fewer number of states (*M* states) are updated in every execution of the policy evaluation algorithm instead of updating all sates. The designated states are determined by the chaotic system. The same procedure is performed in the value iteration method.

In the proposed method, M denotes the number of environmental states that must be updated in each chaotic policy iteration and value iteration.

The algorithm execution time decreases because a smaller number of states are updated in each iteration.

For example, Table 2 shows an output with a chaotic system (after pre-processing) for an environment with 10 states. We can use these numbers if we want to update only 5 states (*M*) at a time. The states 3, 5, 1, 6, and 8 will be updated in the first iteration, and the states 10, 1, 9, 10, and 7 in the second iteration.

Table 2. Few numbers are created by a chaotic system after pre-processing.

1 to 5	3	5	1	6	8
6 to 10	10	1	9	10	7
11 to 15	6	8	2	1	7
16 to 20	8	5	3	10	4

The following steps are taken in the policy iteration method:

First, the following four actions are performed:

1) Start the logistic chaotic system with an appropriate initial value.
2) Put zero for the initial values of all states ($V = 0$).
3) Put the initial policy (π) at random.
4) Insert the number of states (M) that should be updated in every evaluation iteration of a policy

All steps of the policy iteration method are similar to the conventional technique, and only the sub-procedure *PolicyEvaluation* changes to Algorithm 5.

Algorithm 5: Proposed pseudo-code for evaluation of the iterative policy.

Function policyEvaluation (Inputs) **Return** Output
Inputs: π , V(s), M
X// an array of chaotic numbers
i = 0;
Repeat
i = i + 1;
$s = rem(floor((X(i)) \times 10^{14}), S_0);$
$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')];$
Until i <= M
Output: V

In this algorithm, *rem* denotes the remaining integer function, whereas *floor* refers to the integer part of the number, and S_0 shows the number of states in the environment. As a result, s ranges from 0 to S_0-1, and shows an environment state that should be updated. In this algorithm, an M-length array of chaotic numbers is received as an input in each iteration.

In this sub-procedure, the current policy is updated only once for M states of the evaluation environment. In other words, the values of some states are updated with the current policy (this action is done several times for all states in the conventional version of this algorithm); however, the values of the other states are not updated, and the same input value remains. For this purpose, the inplace updating method was applied to the input values of function.

Each time the policy evaluation procedure is called, the next M number is used in the chaotic production series.

Figure 3 shows a flowchart of the proposed policy evaluation method.

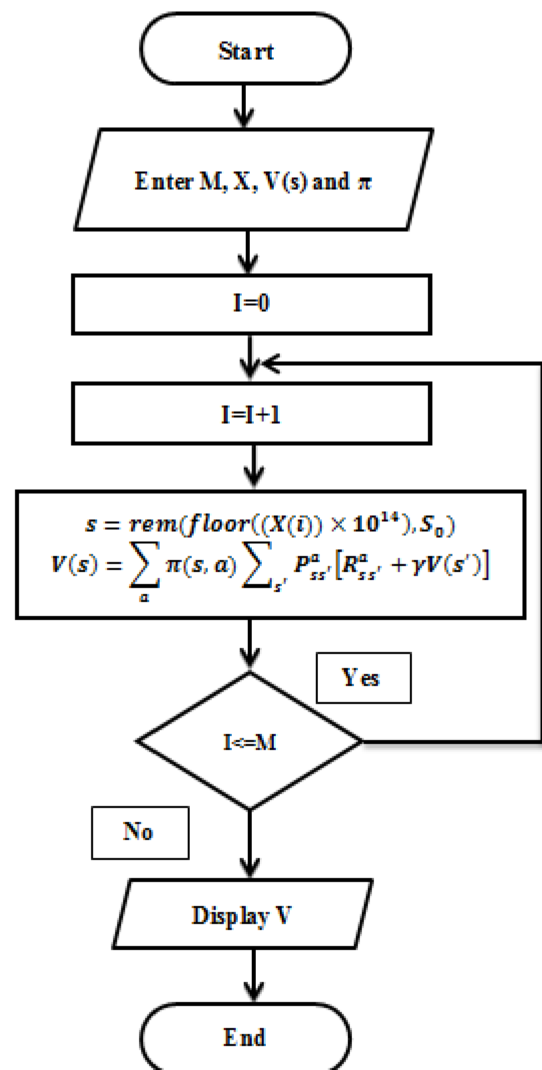

Figure 3. A flowchart of the proposed policy evaluation method.

The value iteration method includes the following steps:

The three following actions are first taken:

1) Start the logistic chaotic system with an appropriate initial value.
2) Put zero in the initial values of all states ($V = 0$).
3) Insert the number of states that should be updated in each value iteration traverse (M).

All steps of the value iteration method resemble the conventional state; however, the new values of M states of the environment are only calculated in the proposed chaotic system instead of obtaining the new values of all states. Algorithm 6 shows the pseudo-code of the proposed method for value iteration.

Algorithm 6: Proposed pseudo-code of the value iteration method.

Function valueIteration (Inputs) **Return** Output
  Inputs: V(s), M
  X // an array of chaotic numbers
  For j = 1 to Number of Element in X
    $C[j] = rem(floor(X(j)) \times 10^{14}), S_0);$
  K = 1;
  Repeat
    ∆ = 0;
    D = C[K..K+M];
    K = K+M;
    For each $s\epsilon D$
      v = V(s);
      $V(s) = \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')];$
      ∆ = max(∆,|v-V(s)|);
    End for
  Until ∆<ε//ε is small positive threshold
  $\pi(s) = \text{argmax}_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$
  Output: π(s)//π(s) ϵA(s): ∀sϵS

The proposed algorithms emphasize the efficiency of chaos with no intelligence for the usefulness evaluation of any states. Based on the features of chaos in visiting different points of the problem state space, the chaotic numbers are mapped onto some states of the environment. After that, only the designated states are updated instead of updating all states.

5. Computational Results

The proposed method was implemented and tested in MATLAB R2018a running on Windows 8 operating on a system with 4 GB of RAM and an Intel Core i5 processor.

Figure 4 shows some samples of the environment for testing the proposed method. In these environments, every cell corresponds to a state. Every cell includes for possible actions, i.e. moving up, moving left, moving right, and moving down. An agent moves definitively in the designated direction, and enters the cell located in that direction near the current cell. The actions that make the agent leave the network or hit an obstacle will make no changes in the agent's success; however, they make the agent face a penalty of -1 in the environment unless the performed action helps the agent reach the target. In this case, it will receive a reward of +1. An example of solving these problems is also shown in Figure 4.

In this figure, there are 399, 47, 423, 44, 724, 139, and 134 accessible states in a, b, c, d, e, f, and g, respectively. This number does not include obstacles.

In addition to the environments presented in Figure 4, another environment called h includes 2500 accessible states. Having no obstacles, it is a 50*50-grid environment, where the final cell on the lower right-hand corner is the target cell.

In Algorithm 1, ε was put 0.01, whereas it was put zero in Algorithms 4 and 6.

The logistic chaotic equation has two parameters λ and an initial value. According to [23], if λ equals 3.9999, the system shows appropriate chaotic attributes; therefore, this value was used in the tests. Several initial values and λ's were used in [24] in order to achieve the desired goals. They resulted in a proper efficiency; thus some of those parameters were also used in this work. According to [19], the closer λ is to 4, the more appropriate chaotic attributes the system shows. It is also possible to discard the initial production numbers and use the subsequent numbers (for example, do not use the first 3000 numbers) to eliminate the transient state of the chaos system.

Table 3 presents the results. Accordingly, each output results from the mean of five consecutive executions in the same conditions. The proposed idea was tested with different parameters. However, given the extensiveness of the logistic system parameters, it is possible to obtain better results. Moreover, S denotes the total number of the visited states until the algorithm convergence. In the policy iteration method, it is necessary to update the existing states in the two phases of policy iteration and policy improvement. In addition, t shows the necessary time until convergence. It also includes generating the random or chaotic numbers, whereas X_0 and M denote the initial value of the logistic system and the number of states updated in policy evaluation or value iteration, respectively.

Since the *max* operator always returns the first value in MATLAB if there are multiple maximum values, a very small random number is added temporarily to its inputs to randomly select one of the inputs equal values.

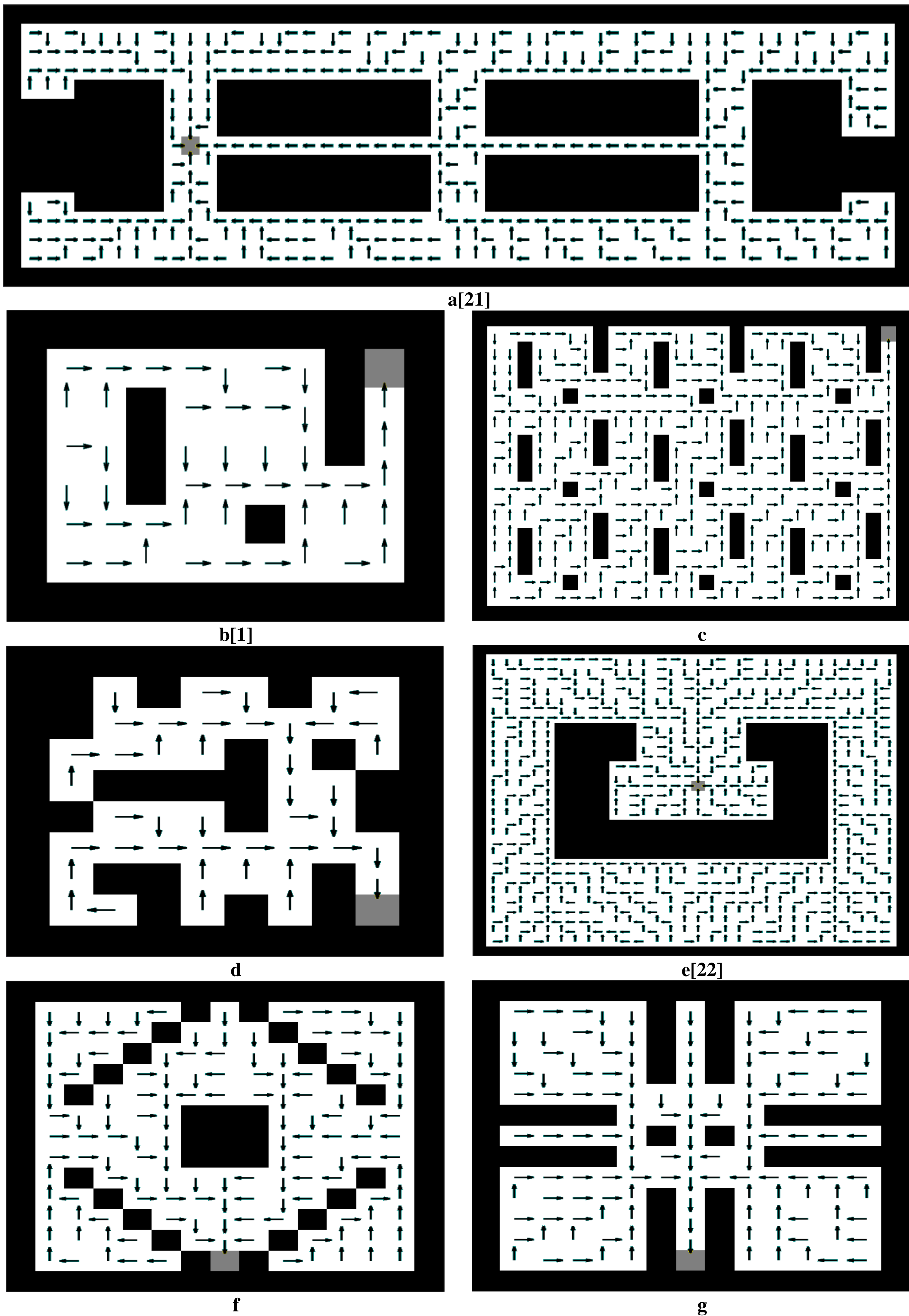

Figure 4. Environments used for testing the proposed method (an example of solving these environments can also be seen).

According to Table 3, the *rand* function was used in MATLAB to generate random numbers in order to determine the superiority of chaos to randomness. The resultant random numbers were then used instead of the chaotic numbers. Evidently, if the same idea is to be implemented through random numbers, the problem will fail to become convergent in many cases. If the problem becomes convergent, a longer period of time than chaos is spent. In addition, it is impossible to repeat the state in which the solution is found randomly (nature of randomness and probabilities). In other words, there might be some variations in the results of a random state if it is repeated. However, the chaos (when the initial parameters are determined) always repeats the previously generated numbers; therefore, it yields the exact same results.

In a random state, the convergence criterion is to reach convergence in all five consecutive states with the given value of M. In fact, it would be insufficient to reach convergence in only some of the executions.

This idea managed to reduce the execution time and decrease the number of states visited to find the optimal solution. The chaotic value iteration method yielded the best performance in all of the tests.

In the use of chaos, M can be put any number to measure convergence and the algorithm execution time. However, since only M states are updated in every iteration of the proposed algorithm, it is not preferable to use a value exceeding the number of environment states, for it is possible to update all sates at once. Moreover, the designated M is a value by which the convergence is reached, and the execution time is reduced. There are also other values by which the convergence is achieved; however, they fail to reduce the execution time less than the conventional method.

In comparison to the standard policy iteration and value iteration methods, only two parameters of chaotic system and M are added. Accordingly, the purpose of changing the chaotic system parameter is to acquire a better chaotic attribute, which is of little importance in view of the present research objective. Therefore, having a new environment with N states, we only require to find the M value that is theoretically an integer as $0 < M < N$. Evidently, an M value that can create convergence in a shorter period of time than the conventional state must not be near 0 or N. Thus the only effective parameter is M.

Table 3. Comparing different algorithms in terms of efficiency in solving environments of Figure 4.

Environment \ Algorithm	Policy iteration	Random policy iteration	Chaotic policy iteration	Value iteration	Random value iteration	Chaotic value iteration
a	t = 3.1338 S = 96873.0	t = 2.7491 S = 40858.0 M = 370	t = 2.5973 S = 35934 M = 280 X_0 = 0.9734	t = 0.5176 S = 18706	t = 0.4827 S = 16112.0 M = 380	**t = 0.3741** S = 12420 M = 180 X_0 = 0.15
b	t = 0.1107 S = 5529.2	-	t = 0.0837 S = 1562 M = 25 X_0 = 0.15	t = 0.0415 S = 736	-	**t = 0.0334** S = 490 M = 35 X_0 = 0.15
c	t = 1.0659 S = 95878.0	-	T = 0.8518 S = 36100 M = 300 X_0 = 0.15	t = 0.5179 S = 18568	-	**t = 0.3535** S = 11160 M = 360 X_0 = 0.52
d	t = 0.2802 S = 6054.4	-	t = 0.1749 S = 1216 M = 33 X_0 = 0.15	t = 0.1064 S = 688	-	**t = 0.0899** S = 462 M = 33 X_0 = 0.9734
e	t = 1.6967 S = 180750	t = 1.6191 S = 79973 M = 700	t = 1.4885 S = 71983 M = 625 X_0 = 0.9734	t = 1.0432 S = 37596	t = 0.8339 S = 29394.0 M = 690	**t = 0.7492** S = 27000 M = 540 X_0 = 0.15
f	t = 0.3558 S = 23380	-	t = 0.2109 S = 6789 M = 80 X_0 = 0.15	t = 0.1071 S = 3197	-	**t = 0.0862** S = 2240 M = 70 X_0 = 0.52
g	t = 0.2629 S = 23155	-	t = 0.1689 S = 5525 M = 87 X_0 = 0.15	t = 0.0822 S = 2546	-	**t = 0.0752** S = 1980 M = 90 X_0 = 0.9734
h	t = 10.3441 S = 732207	t = 9.4365 S = 495260 M = 2300	t = 9.0920 S = 469900 M = 2200 X_0 = 0.52	t = 5.9810 S = 247401	-	**t = 2.6922** S = 105400 M = 1700 X_0 = 0.52

According to Table 3, the chaotic method reduced the number of visited states and the time required to find the optimal solution.
The models focused on grid environments, in any of which the tests can be conducted. Moreover, some of them are standard environments employed in order to examine new respective methods. This practice explicitly outperformed the conventional methods of policy iteration, value iteration, and random methods. This manuscript aims to offer a solution to the problems of conventional methods; hence, the chaotic attribute was used for the first time ever.

6. Conclusions

In this paper, we proposed a method for increasing the speed and efficiency of the chaotic dynamic programming methods in certain environments. It was also recommended to use chaotic equations in order to solve the problem of traversing the entire sets of states in dynamic programming. In this method, only a few of the states are updated in every policy evaluation period or every value iteration. These updated states were proposed by chaos. According to the simulation results, the use of chaos can decrease the number of visited states until convergence is reached. It can also decrease the time spent on this process. The test results show the efficiency and speed of the proposed method. In other words, it took a shorter time than the conventional methods to solve the problem.
The results obtained concern the use of a logistic chaotic system; however, they might improve if other chaotic systems were used.
The results were obtained from the implementation of the proposed method in grid environments; therefore, further studies must be conducted to prove its efficiency in other environments.

References

[1] V. Derhami, F. Alamian Harandi, and M.B. Dowlatshahi, *Reinforcement Learning*. Yazd, Iran, Yazd University Press, 2017.

[2] A.G. Barto, S.J. Bradtke, and S.P. Singh, "Learning to act using real-time dynamic programming", *Artificial Intelligence*, Vol. 72(1), pp. 81–138, 1995.

[3] B. Bonet and H. Geffner, "Labeled RTDP: Improving the Convergence of Real-time Dynamic Programming", *In Proc. 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, 2003, pp. 12–21.

[4] H.B. McMahan, M. Likhachev, and G.J. Gordon, "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees", *In Proceedings of the 22nd International Conference on Machine Learning*, New York, 2005, pp. 569-576.

[5] S. Sanner, R. Goetschalckx, K. Driessens, and G. Shani, "Bayesian real-time dynamic programming", *In Proceedings of the 21st International Joint Conference on Artificial Intelligence,* Pasadena, California, USA, 2009, pp. 1784-1789.

[6] S. Schmoll and M. Schubert. "Dynamic resource routing using real-time dynamic programming", *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, Stockholm, Sweden, 2018, pp. 4822-4828.

[7] P. Dai, M.D.S. Weld, and J. Goldsmith, "Topological value iteration algorithms", *journal of Artificial Intelligence*, Vol. 42(1), pp. 181–209, 2011.

[8] W. Reis, L. Barros, and K. Delgado, "Robust topological policy iteration for infinite horizon bounded Markov Decision Processes", *International Journal of Approximate Reasoning*, Vol. 105, pp. 287–304, 2019.

[9] D. Wingate and K. D. Seppi, "Prioritization methods for accelerating MDP solvers", *In Journal of Machine Learning Research*, Vol. 6, pp. 851-881, 2005.

[10] A. Khademi, "A Novel Method for Improving Value Iteration in Dense Markov Decision Process", M.S. thesis, Dept. Computer., Yazd Univ., Yazd, 2016.

[11] E.A. Hansen and S. Zilberstein, "Lao*: A heuristic search algorithm that finds solutions with loops", *Artificial Intelligence*, 129(1-2), pp. 35–62, 2001.

[12] B. Bonet and G. Hector, "Action selection for MDPs: Anytime AO* *vs.* UCT", *In AAAI Conference on Artificial Intelligence*, 2012, Toronto, Ontario, Canada, pp. 1749-1755.

[13] H. Khodadadi and A. Zandvakili, "A New Method for Encryption of Color Images based on Combination of Chaotic Systems", *Journal of Ai and Data Mining*, Vol. 7(3), pp. 377-383, 2019.

[14] H. Khodadadi and O. Mirzaei, "A stack-based chaotic algorithm for encryption of colored images", *Journal of AI and Data Mining*, Vol. 5(1), pp. 29-37, 2017.

[15] X. Chai, Y. Chen, and L. Broyde, "A novel chaos-based image encryption algorithm using DNA sequence operations", *Optics and Lasers in Engineering*, Vol. 88, pp. 197-213, 2017.

[16] J.S.A.E. Fouda, J.Y. Effa, S.L. Sabat, and M. Ali, "A fast chaotic block cipher for image encryption", *Communications in Nonlinear Science and Numerical Simulation*, Vol. 19(3), pp. 578-588, 2014.

[17] E.N. Lorenz, "Deterministic Non-periodic Flow", *Journal of the Atmospheric Sciences*, Vol. 20(2), pp.130-141, 1963.

[18] G. Chen and T. Ueta, "Yet another chaotic attractor", *International Journal of Bifurcation and Chaos*, Vol. 9(7), pp. 1465-1466, 1999.

[19] A. Kanso and N. Smaoui, "Logistic chaotic maps for binary numbers generations", *Chaos, Solitons and Fractals*, Vol. 40(5), pp. 2557–2568, 2009.

[20] B. Youngchul, "Target Searching Method in the Chaotic Mobile Robot", *In the 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, Salt Lake City, UT, USA, 2004.

[21] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[22] S. Debnath, L. Liu, and G. Sukhatme, "Reachability and differential based heuristics for solving Markov decision processes", *The 18th International Symposium on Robotics Research (ISRR)*, Puerto Varas, Chile, 2017.

[23] N.K. Pareek, V. Patidar, and K.K. Sud, "Image encryption using chaotic logistic map", *Image and Vision Computing*, Vol. 24(9), pp. 926-934, 2006.

[24] L. Wang and H. Cheng, "Pseudo-Random Number Generator Based on Logistic Chaotic System", *Entropy*, Vol. 21(10), 2019.

افزایش سرعت و کارائی روش‌های برنامه‌سازی پویا به کمک آشوب

حبیب خدادادی و ولی درهمی*

دانشکده مهندسی کامپیوتر، دانشگاه یزد، یزد، ایران.

ارسال ۲۰۲۱/۰۲/۰۴؛ بازنگری ۲۰۲۱/۰۷/۰۷؛ پذیرش ۲۰۲۱/۰۸/۱۳

چکیده:

یکی از ضعف‌های عمده روش‌های برنامه‌سازی پویا انجام عملیات در سرتاسر مجموعه حالت‌های یک فرآیند تصمیم گیری مارکوف در هر مرحله به‌روزرسانی است. در این مقاله روشی جدید بر مبنای آشوب برای غلبه بر این مشکل ارائه شده است. ابتدا با دادن مقادیر اولیه یک سیستم آشوبناک شروع شده و با پردازش‌های اولیه، اعداد تولیدی به حالت‌های محیط نگاشت داده می‌شود. در هر پیمایش روش تکرار سیاست، ارزیابی سیاست فقط یک بار انجام می‌شود و تنها تعداد معدودی از حالت‌ها به‌روزرسانی می‌شود که این حالت‌ها توسط سیستم آشوب پیشنهاد می‌شوند؛ در این روش، چرخه ارزیابی و بهبود سیاست تا رسیدن به یک سیاست بهینه در محیط ادامه پیدا می‌کند. همین عمل در روش تکرار ارزش نیز انجام می‌شود و در هر پیمایش، فقط ارزش تعدادی از حالاتی که آشوب پیشنهاد می‌دهد به‌روزرسانی شده و ارزش بقیه حالات بدون تغییر باقی می‌ماند. بر خلاف روش‌های معمولی، در این شیوه تنها با به‌روزرسانی تعداد معدودی از حالت‌هائی که به خوبی توسط آشوب در سرتاسر فضای محیط پخش شده‌اند، می‌توان به حل بهینه رسید. آزمایش‌های انجام شده نشان‌دهنده افزایش سرعت و کارائی روش‌های برنامه‌سازی پویای آشوبناک در بدست آوردن حل بهینه در محیط-های مختلف گریدی است.

کلمات کلیدی: آشوب، برنامه‌سازی پویا، تکرار ارزش، تکرار سیاست، سیستم آشوبناک لاجستیک، یادگیری تقویتی.