An iterative stochastic algorithm based on distributed learning automata for finding the stochastic shortest path in stochastic graphs

Hamid Beigy & Mohammad Reza Meybodi

The Journal of Supercomputing
An International Journal of High-Performance Computer Design, Analysis, and Use

ISSN 0920-8542
Volume 76
Number 7

J Supercomput (2020) 76:5540-5562
DOI 10.1007/s11227-019-03085-0

Springer

The Journal of Supercomputing (2020) 76:5540–5562
https://doi.org/10.1007/s11227-019-03085-0

An iterative stochastic algorithm based on distributed learning automata for finding the stochastic shortest path in stochastic graphs

Hamid Beigy[1] · Mohammad Reza Meybodi[2]

Published online: 27 November 2019

Abstract

In this paper, we study the problem of finding the shortest path in stochastic graphs and propose an iterative algorithm for solving it. This algorithm is based on distributed learning automata (DLA), and its objective is to use a DLA for finding the shortest path from the given source node to the given destination node whose weight is minimal in expected sense. At each stage of this algorithm, DLA specifies edges needed to be sampled. We show that the given algorithm finds the shortest path with minimum expected weight in stochastic graphs with high probability which can be close to unity as much as possible. We compare the given algorithm with some distributed learning automata-based iterative algorithms, a particle swarm optimization-based algorithm, an ant colony-based algorithm, a Q-learning-based algorithm, and an actor–critic-based algorithm for finding the shortest path. Computer experiments show that the proposed algorithm requires fewer edge samples to find the shortest path than the previously introduced DLA-based algorithms.

Keywords Learning automata · Stochastic shortest path · Distributed learning automata

1 Introduction

A weighted graph $G = (V, E)$ consisting a set of nodes V, a set of edges E, and a weight (cost) function $W : E \rightarrow \mathbb{R}^{+}$. Weight of path $\pi = \langle v_0, v_1, \ldots, v_k \rangle$ from a node v_{s} called source node to a node v_{d} called destination node equals to sum of its edges'

✉ Hamid Beigy
beigy@sharif.edu

Mohammad Reza Meybodi
mmeybodi@aut.ac.ir

[1] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

[2] Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran

weight, where $v_0 = v_\mathrm{s}$, $(v_i, v_{i+1}) \in E$, and $v_k = v_\mathrm{d}$, i.e., $W(\pi) = \sum_{i=1}^{k} W\big(v_{i-1}, v_i\big)$. A path from v_s to v_d with minimum weight is called the shortest path, and there are several algorithms for finding such path in deterministic graphs [14]. In some applications, there are uncertainties, which make stochastic graphs a better model. This model has applications in computer networks, emergency service delivery, and transportation, to mention a few. For example, we have different delays (including the queue time in every node) for different links in computer networks. These delays have unknown distribution, and minimum delay routing can be considered as stochastic shortest path problem. A transportation network can be represented as a graph in which nodes represent the intersections, and roads/streets show edges of the graph. In this network, edge lengths can be considered as the travel time or the cost of traveling in which the travel time between two intersections is a random variable with unknown distribution, whose distribution in each street/road depends on the traffic situation on that edge. We have the actual value of travel time for each edge when traveling from that edge. This travel time is sampled form the corresponding distribution and may change in different traveling. This problem is called *route guidance systems (RGS)* in *intelligent transportation systems*, and the goal for solving it is the design of an algorithm for finding the shortest path from the source to the destination. This shortest path between the given source and the given destination is defined as the one with minimum expected travel time for most RGS [18].

We define stochastic graph $G = (V, E, \mathcal{F})$ using a triple consisting of V as a set of nodes, E as a set of edges E, and $\mathcal{F}$ as a probability distribution function specifying the edge weights. We suppose that the weight of an edge (i, j), shown as $W(i, j)$, is modeled using a random variable taking only positive values and $\mathcal{F}$ specifies their joint probability. In undirected stochastic graphs, both directions of each edge have the same weight. We assumed that the weight of different edges is independent random variables specified by the probability distribution $\mathcal{F}$. We also assume that the range of $W(i, j)$ is finite. The probability distribution function $\mathcal{F}$ can be discrete or continuous. In the case of the discrete distribution function, the possible values of each edge weight with their associated probabilities are given, while in the case of the continuous distribution function, only the distribution function with its associated parameters is given for each edge.

The weight of path $\pi = \langle v_0, v_1, \dots, v_k \rangle$ from node v_s to node v_d, denoted by $W(\pi)$ equals to the sum of its edges' weights, where $v_0 = v_\mathrm{s}$ and $v_k = v_\mathrm{d}$, i.e., $W(\pi) = \sum_{i=1}^{k} W\big(v_{i-1}, v_i\big)$. From the fact that the path weight is a random variable, the path with minimum value of edge weights in expected sense is called the shortest path. The problem of finding such a path in stochastic graphs is called *stochastic shortest path problem* (SSPP). We define the expected weight of path π with $\bar{W}(\pi)$ and the shortest path of the graph with π^*. Two different approaches were introduced to solve this. The first approach tries to find *a priori* solution for minimizing the expected weight, and the second approach tries to compute online solutions. Pritsker [44] and Frank [17] analyzed some problems for general stochastic graphs. The stochastic shortest path problem also has been analyzed using the concept of the unique arcs [23] and the uniformly directed cuts [48]. In these references, probabilistic quantities, which are needed by the algorithms, represented as multiple integrals, where their computation is a time consuming process, even for small graphs. To avoid the evaluation of such multiple integrals, a different method is proposed in [31]. This method can be applied only to the stochastic graphs when their edge weights are discrete random variables. Since the exact numerical computation of the weights distribution of paths is computationally difficult, Monte Carlo simulation was used [1, 47]. In [2], state-space partitioning has been used for finding the stochastic shortest path. In the algorithm proposed in [43], when v_{d} is reached the weight of edges become known. Then, dynamic programming methods were used for solving this problem [29, 30]. In [55], SSPP with recourse was studied. In [16], dynamic programming solution was used for solving the mentioned problem when the number of edges of each path is limited. In [41], the weight for edges considered as exponentially distributed random variables and the SSPP was formulated as a linear programming problem. In [53], a network setting was used in which each edge weight is modeled using a random variable with known distribution. In [13], it showed that the SSPP with the limited number of hops, i.e., paths with at most $n-1$ edges, is a NP-Hard problem. Then, they extended the results given in [16] for solving SSPP with the unlimited number of hops. It must be noted that in all the above-mentioned studies, the distributions of edge weight are known in before stating the algorithm. In [33], a particle swarm optimization (PSO)-based iterative algorithm was given for solving SSPP, which did not know the probability distribution of edge weights.

In [51], the stochastic shortest paths with no dead-end nodes are considered and an infinite-horizon dual optimization criterion is proposed. The shortest path in the Markov decision process (MDP), in which short-sightedness is used only to determine whether a state should be labeled as solved or not, is studied by Pineda et al. [42]. Guillot and Stauffer proposed a new framework for solving SSPP in finite state and action spaces in context of MDP [19]. The stochastic shortest path with correlated link travel times is considered in [56]. It is assumed that edge and path lengths are, respectively, fitted by lognormal and normal distribution. The authors of [12] proposed algorithms to find the most reliable path in large-scale road networks. It is assumed that the path lengths follow normal distributions and link lengths are mutually independent.

Let the probability that path π is the shortest be denoted by $\mathbb{P}[\pi]$. One iterative approach for computing $\mathbb{P}[\pi]$ is to use distributed learning automata (DLA) in which multiple automata cooperate to solve the given problem. In [8], a DLA-based iterative algorithm was presented for finding the stochastic shortest path. To compute $\mathbb{P}[\pi]$, a DLA is constructed from the given graph, and its action probability vectors are updated until the shortest path is found. In [28], an extended version of DLA was proposed in which each LA has an activity level and only the LA with highest activity level can perform an action on the environment. The experimental results reported in [28] shows that this algorithm requires a smaller number of samples from the graph than the algorithm given in [8]. Vahidipour et al. proposed a framework that generalizes different frameworks for fusion of LAs and Petri nets for solving different graph-based problems [54]. The above-mentioned algorithms are different from each other from the respect to (1) the learning algorithm of LAs, (2) the updating algorithm for learning rate α_k, (3) the activation strategy of LAs, (4) the method for avoiding cycles, and (5) the direction in which LAs update their action probability vectors.

Algorithms for solving SSPP based on the learning time of edges' weights can be classified into three categories [43]: (1) the weight values of edges are estimated before traversing the graph [17], (2) the weight values are estimated progressively when traversing the graph [32, 43], and (3) the weight values of edges are never learned or become known. In stochastic graphs, we should select a path which is shortest with respect to the expected values of edge weights and the third model is used in this paper. In this paper, we propose a new algorithm, which is based on DLA, and requires fewer samples from edges for producing the shortest path. In this algorithm, the evaluation of the environment is different from the algorithm given in [8]. It proved that the given algorithm can find the shortest path with a high probability. The given algorithm does not need knowledge of probability distribution of weights of edges, which is an advantage of this algorithm. The experimental results show this algorithm is superior over some other iterative algorithms, which implies that it needs the smaller number of samples.

The remaining section is organized as: the proposed algorithm and the study of its behavior are given in Sect. 2. The results obtained from experiments on some benchmark graphs are given in Sect. 3 and in Sect. 4 concluding remarks are given.

2 The proposed algorithm

The learning in learning automata (LA) is finding the best action from its allowable action set. At time k, automaton chooses one action, say action α_i and gives it to an environment, which is random and gives a grade to the selected action by generating a stochastic response $\beta(k) \in \{0, 1\}$ that is given as the input to the automaton. Action α_i penalized with penalty probability c_i by getting a signal $\beta(k) = 1$ and rewarded with reward probability d_i $(d_i = 1 - c_i)$ by getting a signal $\beta(k) = 0$. Then, the learning algorithm of the automata updates the action probability vector $\underline{p}$, where $p_i(k)$ shows the probability of choosing action α_i at time k. The penalty probabilities are unknown initially, and the automaton's goal is finding the minimum penalty probability action. Linear reward-inaction learning algorithm (L_{R-I}) updates vector $\underline{p}$ using

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)] & \text{if } i = j \\ p_j(k)(1 - a) & \text{if } i \neq j \end{cases} \tag{1}$$

when action α_i gets reward and $\underline{p}$ is unchanged when action α_i gets penalty [37]. Parameter $a \in (0, 1)$, which is called *learning rate*, specifies the amount of changes in the action probabilities. Learning automata were used successfully in problems such as data networks and telephone routing [38, 49], finding solution intractable problems [4, 8, 40], capacity assignment [39], neural network engineering [3, 5, 11, 24–26], social networks [35, 36] design of Bayesian optimization algorithms [34], cellular networks [6, 7, 9, 10], intelligent random walks [46], to mention a few.

LAs have limited capability, and their capability can be increased by using interconnection of multiple LAs in the form of a tree, a mesh, an array, etc. These automata collectively cooperate for solving a problem. DLA is an interconnected model of LAs, where the interaction is a directed graph. Each node of this graph is an automaton, and outgoing edges of every node are actions of that automaton. In DLA, only one LA acts at any time instant and its selected action determines the next active automaton. Hence, activation passes from one LA to another one and continues until the last LA was reached. When A_{s} activated, the sequence of selected actions of active LAs is given to the environment, and its response is used for updating their action probability vectors by using the given learning algorithm. DLA are used successfully for solving SSPP [8, 28], grid resource discovery [20], finding maximum clique [45], and link prediction in social networks [35], to mention a few.

In what follows, we give an iterative algorithm, which is based on DLA and finds the shortest path from v_{s} to v_{d} in a stochastic graph. The given algorithm has an advantage that it does not need knowledge of probability distribution of edges' weights. In this algorithm, the role of environment for the DLA is played by the stochastic graph and every path from v_{s} to v_{d} is an action of DLA. The weight of this path is used by the environment to generate the reinforcement signal. This signal, depending on whether it is *reward* or *punishment*, is used for updating the actions probabilities of the activated LAs.

The high-level description of the given algorithm is shown in algorithm 1, which is described in what follows. First, a DLA created. The graph of this DLA is isomorphic to the input graph. After that, N random paths from v_{s} to v_{d} are traversed and the average value of the weight of these N random paths stored in $\bar{C}_0$. This avoids the premature convergence to a non-shortest path. In our experiments, we used only 10 random generated paths. Then, at stage k, a path starting from v_{s} is found according to the DLA in the following way. Automaton A_{s} chooses an action α_m, which activates automaton A_m. This process continues until A_{d} is reached or for some reasons none of actions cannot be selected, or the number of edges in the chosen path is greater than or equal to $|V|$. When a path from A_{s} to A_{d} is found, its weight denoted by $W(\pi)$, and the average weight of the traversed paths, $\bar{C}_k$, up to stage k are computed, where π is the traversed path. The weight of a path π is found by sum of weights of its constituting edges. These edges' weights are sampled from their corresponding distribution. Then, $\bar{C}_k$ is compared with a quantity called *dynamic threshold*, denoted by T_k, and the result of this comparison is used to update the value of dynamic threshold and also the action probability vectors of activated LAs are updated. In this algorithm, we define the dynamic threshold as

$$T_k = \begin{cases} \infty & \text{if } k = 0 \\ \min\{\bar{C}_0, \ldots, \bar{C}_{k-1}\} & \text{if } k > 0 \end{cases} \tag{2}$$

When $\bar{C}_k < T_k$, then the dynamic threshold is set to $\bar{C}_k$ and the selected action of DLA gets reward. When $\bar{C}_k \geq T_k$, then the dynamic threshold remains unchanged, and the selected action of DLA gets penalty. The L_{R-I} algorithm was used for updating the action probability vectors from A_{d} to A_{s}. In the updating phase, learning rates of LAs A_{π^j} (for $j = 2, 3, \ldots$) along the path π updated using

$$a_{\pi^j}(k) = \frac{a_{\pi^{j-1}}(k)}{p_{\pi^j}^{\pi^{j-1}}(k+1)}, \tag{3}$$

where $a_{\pi^j}(k)$ is the learning rate of *j*th LA along the traversed path at stage *k*, $p_{\pi^j}^{\pi^{j-1}}$ is the probability of selecting action α_{π^j} by LA $A_{\pi^{j-1}}$, and $a_{\pi^1(k)} = a$ for all $k \geq 1$. Equations (1) and (3) guarantee that $a_{\pi^j}(k) < 1$ for all k and for all j when $a < 1$. These equations also imply that $a_{\pi^1}(k) < a_{\pi^2}(k) < a_{\pi^3}(k) < \cdots$ for all k. We can conclude that the updating the learning rate of automata done according to equation (3) causes those LAs which are farther from A_{s} converge more quickly than those LAs which are nearer to A_{s}. This leads to a higher speed of convergence for the algorithm. The hierarchical systems of learning automata were using the same approach for updating the learning rate [52]. The process of choosing a path from A_{s} to A_{d} is repeated until the stopping criteria are satisfied. When the algorithm reaches its stopping criteria, the last chosen path is with high probability the shortest one. The stopping criteria of the algorithm is defined as at least one of the two conditions is satisfied: (1) if the *path probability* and denoted by $\mathbb{P}(\pi)$, which is the probability of selecting the given path, exceeds P_{pop}, or (2) the number of iterations becomes greater than threshold K. In order to have a path without any loop, the algorithm only activates every node along a path at most once. For implementing such a rule, when an LA chooses action α_j, then α_j will be disabled from action-sets of all inactivated LAs. When traveling again from destination node to source node, we enable all actions, which were disabled.

Algorithm 1 The proposed algorithm for solving stochastic shortest path problem.

```
function SHORTESTPATH($G, v_s, v_d, N, P_{pop}, K$)
    Build a DLA from input graph $G$
    Travers $N$ random paths from $v_s$ to $v_d$.
    Let $\bar{C}_0$ be the average weight of these paths.
    Let $k \leftarrow 1$ and $T_1 \leftarrow \infty$
    repeat
        Find a path $\pi$ starting from node $v_s$.
        Let $W(\pi)$ be the weight of path $\pi$.
        if ($v_d$ is reached) then
            $\bar{C}_k \leftarrow \bar{C}_{k-1} + \frac{1}{k}\left[W(\pi) - \bar{C}_{k-1}\right]$.
            if ($\bar{C}_k < T_k$ ) then
                $T_{k+1} \leftarrow \bar{C}_k$
                Reward actions constituting path $\pi$.
            else
                $T_{k+1} \leftarrow T_k$
            end if
            $k \leftarrow k + 1$
        end if
    until ( $\mathbb{P}(\pi) \geq P_{pop}$ or $k \geq K$ )
    return $\pi$.
end function
```

Theorem 1 shows that the proposed algorithm is given in algorithm 1 solving the SSPP with a high probability. The method used for the proof is very similar to the method given in [22, 27, 52] in the context of the analysis of LA operating in non-stationary environments. Before giving the main result, we give and prove two lemmas.

Lemma 1 *We denote $q_i(k)$ as the probability of choosing path π_i at iteration k and also denote $\underline{q}(k) = \left(q_1(k), q_2(k), \dots\right)^{\mathrm{T}}$ as the probability of choosing different paths in the graph. Define random variable $Y_i(k)$ as the number of times path π_i was chosen up to time k. When $q(k)$ evolved using algorithm 1, then for every $\delta = \delta(M, K) > 0$, there exists a δ, $a^* \overline{\in} (0, 1)$, $M > 0$, and $k_0 < \infty$ such that for all $k > k_0$ and for all $a \in (0, a^*)$, the following inequality holds*

$$\mathbb{P}\left[Y_i(k) \geq M\right] > 1 - \delta \qquad \forall i = 1, 2, \dots. \tag{4}$$

Proof We must prove that

$$\mathbb{P}\left[Y_i(k) \geq M\right] > 1 - \delta, \tag{5}$$

and equivalently $\mathbb{P}\left[Y_i(k) \leq M\right] \leq \delta$. Let us to define another random variable $Z(\pi, k) = 1$ when path π is chosen at time k and $Z(\pi, k) = 0$, otherwise. Thus, we have $\mathbb{E}[Z(\pi_i, k)] = q_i(k)$. Hence,

$$\begin{aligned} \operatorname{Var}\left[Z(\pi_i, k)\right] &= \mathbb{E}[Z(\pi_i, k)^2] - \mathbb{E}[Z(\pi_i, k)]^2 \\ &= q_i(k)[1 - q_i(k)]. \end{aligned} \tag{6}$$

Since for all $s \neq t$, events $\{Y_i(k) = s\}$ and $\{Y_i(k) = t\}$ are mutually exclusive, then $Y_i(k) = \sum_{s=1}^{k} Z(\pi_i, s)$. From Eqs. (1) and (3), it follows that $q_i(k+1) \geq q_i(k)(1-a) \geq q_i(0)(1-a)^k$, where a is the learning rate of automaton A_{v_s}. Hence, $\mathbb{E}[Y_i(k)]$ becomes

$$\begin{aligned} \mathbb{E}[Y_i(k)] = \sum_{s=1}^{k} \mathbb{E}[Z(\pi_i, k)] &\geq \sum_{s=1}^{k} q_i(0)(1-a)^s, \\ &= q_i(0) \frac{(1-a) - (1-a)^{k+1}}{a}. \end{aligned} \tag{7}$$

Now, we have

$$\begin{aligned} \lim_{a \to 0} \frac{(1-a) - (1-a)^{k+1}}{a} &= k, \\ \lim_{a \to 1} \frac{(1-a) - (1-a)^{k+1}}{a} &= 0. \end{aligned} \tag{8}$$

So, there exists a learning rate $a(\pi_i, k)$ for path π_i at stage k such that for all $a < a(\pi_i, k)$

$$\mathbb{E}\left[Y_i(k)\right] \geq (k-1) \times q_i(0). \tag{9}$$

Again, since random variables $\{Z(\pi_i, k)\}$ are uncorrelated, we have

$$\begin{aligned}\mathrm{Var}\big[Y_i(k)\big] &= \mathrm{Var}\left[\sum_{s=1}^{k} Z(\pi_i,k)\right]\\ &= \sum_{s=1}^{k} q_i(s)[1-q_i(s)], \\ &\le \sum_{s=1}^{k} q_i(s) = \mathbb{E}[Y_i(k)].\end{aligned} \tag{10}$$

Using the Chebyshev inequality, we obtain

$$\begin{aligned}\mathbb{P}\big[Y_i(k) > M\big] &= 1 - \mathbb{P}\big[Y_i(k) \le M\big]\\ &\ge 1 - \mathbb{P}\big[\big|Y_i(k) - \mathbb{E}[Y_i(k)]\big| \ge \mathbb{E}[Y_i(k)] - M\big]\big]\\ &\ge 1 - \frac{\mathrm{Var}\big[Y_i(k)\big]}{\big(\mathbb{E}[Y_i(k)] - M\big)^2}\\ &= 1 - \frac{\mathbb{E}[Y_i(k)]}{\big(\mathbb{E}[Y_i(k)] - M\big)^2}.\end{aligned} \tag{11}$$

Inequality (9) shows that $\mathbb{E}[Y_i(k)]$ increases linearly with k. For a fixed M, for all $k > k_o(\pi_i,k)$, and $a < a(\pi_i,k)$, there exists a $k_o(\pi_i,k)$ and a $a(\pi_i,k)$ such that

$$\mathbb{P}[Y_i(k) \ge M] \ge 1 - \delta. \tag{12}$$

Now considering all paths from v_s to v_d, there exists $a^* \in (0,1)$ and $k_o < \infty$ such that

$$\begin{aligned}k_o &= \max_{\pi}\{k_o(\pi,k)\}\\ a^* &= \min_{\pi}\{a(\pi,k)\}.\end{aligned} \tag{13}$$

Hence,

$$\mathbb{P}\big[Y_i(k) > M\big] \ge 1 - \delta, \tag{14}$$

for all $a \in (0, a^*)$ and for all $k > k_o$. □

Lemma 2 *Let path π_i is chosen at iteration k in a stochastic graph with unique shortest path. Let $c_i(k) = \mathbb{P}[\bar{C}_k \ge T_k]$, $c_i^* = \lim_{k\to\infty} c_i(k)$, and path π_i is chosen $Y_i(k)$ times up to iteration k. Then for any $\epsilon > 0$ and $\delta \in (0,1)$, the following inequality holds*

$$\mathbb{P}\big[\big|c_i^* - c_i(k)\big| > \epsilon\big] < \delta,$$

for all $k > k_o(\epsilon,\delta)$, for all $a < a^*(\epsilon,\delta)$, and for all $i = 1,2,\dots,r$.

Proof Consider two smallest penalty probabilities from $\{c_1^*, \dots, c_r^*\}$ and denote their difference by Δ. Then, $\Delta > 0$, because of the uniqueness of the shortest path. Let X_k

be the indicator function such that $X_k = 1$ when at stage k, path π_i was chosen and actions constituting path π_i are rewarded. Hence, by using the weak law of large numbers, which states that for a given $\delta > 0$, there is $k_i(\varDelta, \delta) < \infty$ in such a way when path π_i is chosen at least $k_i(\varDelta, \delta)$ times (i.e., $k_i(\varDelta, \delta) < Y_i(k)$), we have

$$\mathbb{P}\left[\left|c_i^* - c_i(k)\right| < \frac{\varDelta}{2}\right] > 1 - \delta. \tag{15}$$

From definition of $\varDelta$, for all $j \neq i$ such that $\min_l\{Y_l(k)\} > M$, where $M = \max_l\{k_l(\varDelta, \delta)\}$, we have

$$\mathbb{P}\left[c_j(k) > c_i(k)\right] > 1 - \delta. \tag{16}$$

By Lemma 1, we know that we can define k_o and a^* in such a way that

$$\mathbb{P}\left[\min_i\{Y_i(k)\} > M\right] > 1 - \delta, \tag{17}$$

for all $k > k_o$ and all $a \in (0, a^*)$. Thus if each path is chosen more than M times, each $c_i(k)$ is in a $\varDelta/2$–neighborhood of c_i^* with an arbitrary high probability. □

Theorem 1 *Let the probability of choosing action α_i at iteration k, which equals to the probability of traversing path π_i be denoted by $q_i(k)$ and also let be $\underline{q}(k) = \left(q_1(k), q_2(k), \ldots, q_r(k)\right)^{\mathrm{T}}$, where r is the number of different paths from v_{s} to $\overline{v}_{\mathrm{d}}$. When $\underline{q}(k)$ updated using the given algorithm, then for any value of $\epsilon > 0$, there is a learning rate $a^* \in (0, 1)$ in such a way that for all values of $a \in (0, a^*)$, we have*

$$\mathbb{P}\left[\lim_{k\to\infty} q_l(k) = 1\right] \geq 1 - \epsilon,$$

*where l shows that l*th *path, which is denoted by π^*, is the shortest one.*

Proof The proof of this Theorem consists of the following three steps:

1. In the first step, the probability of choosing path π_i at iteration k denoted by $q_i(k)$ (for $i = 1, \ldots$) are calculated in terms of the action probabilities of the corresponding LAs.
2. In the second step, it is shown that for enough large values of k, $q_l(k)$ is a sub-Martingale process.
3. In the third step, the convergence of the algorithm will be shown using Martingale convergence theorems.

□

This theorem states that algorithm 1 finds the shortest path with a high probability, which can be close to unity as desired. The complete details of the proof of Theorem 1 depend on the specific graph. The detail of the proof follows the general framework that is given in [8].

3 Experiments

We used the computer experiments for evaluation of the given algorithm and compared with two reported DLA-based iterative algorithms [8, 28], one PSO-based algorithm [33], an algorithm based on ant colony, an algorithm based on Q-learning, and an algorithm based on actor–critic. The following four stochastic graphs, which are borrowed from [2], are used to evaluate the given algorithm.

- The first graph is shown in Fig. 1. This graph has 10 nodes and 23 edges. The source node has label 1 and the destination node has label 10. The label of each edge is the mean value of weight distribution for the corresponding edge, which has exponential distribution. For instance, the weight of edge (1, 2) has the exponential distribution with the mean of 12.
- The second graph is shown in Fig. 1 and is the same as graph 1 but with different edge weight distribution as given in Table 1. For this graph $v_s = 1$, and $v_d = 10$.
- The third graph is shown in Fig. 1 and is the same as graph 1 but with different edge weight distribution given in Table 2. For this graph $v_s = 1$, and $v_d = 10$.
- The fourth graph is shown in Fig. 2 and the weight distribution of edges is given in Table 3. This graph has 15 nodes and 42 edges. For this graph, the source node has label 1 and the destination node has label 15.

Tables 1, 2, and 3 show the distribution of edge weights for graphs 2, 3, and 4, respectively. The second column of Tables 1, 2, and 3 shows weight values of each edge, and the third column shows the distribution of weight values for that edge. For instance in Table 1, edge (1,2) has weight 7, 7.3, or 9.4 with probabilities 0.2, 0.5, or 0.3, respectively.

Algorithms are run 100 times on the four graphs, and the results are given in Tables 4 and 5. The results of each algorithm are given in two tables, part (a) and part (b). The average number of iterations for runs that find the shortest path correctly runs (AVI) and the percentage of runs that find the shortest path correctly (PC) are given in the part (a). For instance, the last row in part (a) of Table 4

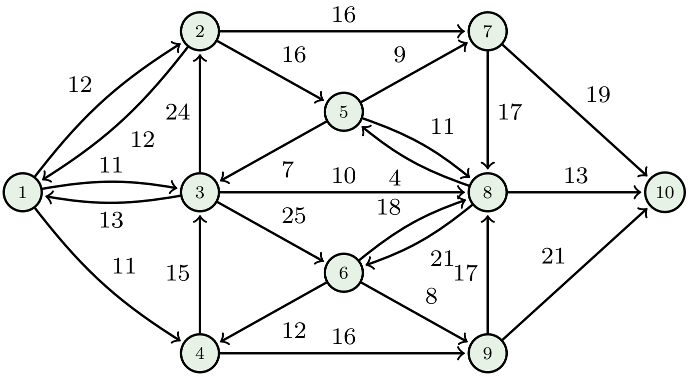

Fig. 1 Graph 1

Table 1 Weight distribution of graph 2 (Fig. 1)

Edge	Weights					Probabilities				
(1,2)	7.0	7.3	9.4			0.2	0.5	0.3		
(1,3)	2.5	3.5	8.2			0.5	0.4	0.1		
(1,4)	4.2	4.8	6.1			0.2	0.3	0.5		
(2,5)	2.6	3.1	5.5	8.8	9.0	0.1	0.2	0.4	0.2	0.1
(2,6)	5.8	7.0	9.5			0.3	0.3	0.4		
(3,2)	1.5	7.3				0.4	0.6			
(3,7)	6.5	7.4	7.5			0.4	0.5	0.1		
(3,8)	5.9	7.2	9.8			0.6	0.3	0.1		
(4,3)	2.1	3.2	8.5	9.8		0.3	0.2	0.3	0.2	
(4,9)	8.9	9.6				0.7	0.3			
(5,7)	3.2	4.8	6.7			0.2	0.2	0.6		
(5,10)	6.3	6.9				0.5	0.5			
(6,3)	6.6	8.5	9.8			0.8	0.1	0.1		
(6,5)	0.6	1.5	3.9	5.8		0.1	0.4	0.3	0.2	
(6,7)	0.2	4.8				0.4	0.6			
(7,6)	6.1	6.3	8.5			0.2	0.3	0.5		
(7,8)	1.6	1.8	4.0	5.2		0.2	0.3	0.3	0.2	
(7,10)	1.6	3.4	7.1			0.1	0.5	0.4		
(8,4)	9.0	9.6				0.5	0.5			
(8,7)	2.1	4.6	8.5			0.3	0.4	0.3		
(8,9)	1.7	4.9	5.3	6.5		0.1	0.4	0.4	0.1	
(7,9)	0.3	3.0	5.0			0.1	0.4	0.5		
(9,10)	0.6	1.2	5.4	6.6		0.1	0.1	0.3	0.5	

shows that for learning rate $a = 0.001$, 99% of runs on graph 1 find the shortest path correctly and each run on average traverses 8402 paths of graph 1. Part (b) of each Table shows the total number of samples (TS) taken by the algorithm from the edges and the number of samples taken from the shortest path edges (SPS). For instance, the last row in part (b) of Table 4 shows that for learning rate $a = 0.001$, we require 36,753 samples from the edges of graph 1 out of which we require 10,739 samples from the shortest path edges.

All algorithms stop when the path probability for the current path is higher than P_{POP}, which is a pre-specified threshold. The algorithms also stop if the path probability cannot reach P_{POP} within a specific number of iterations. The values of these two thresholds depend on the input graph. For the simulation, we set $P_{\text{POP}} = 0.9$ and $K = 300,000$.

Careful inspection of these Tables reveals the fact that the given algorithm requires fewer iterations for convergence (fewer number of samples from the graph edges) in comparison with the algorithm reported in [8]. Note that if we choose $a = 0.0002$, algorithm 1 while maintaining the same rate of convergence requires a fewer number of iterations than needed by the algorithm reported in [8].

Table 2 Weight distribution of graph 3 (Fig. 1)

Edge	Weights				Probabilities			
(1,2)	3	5.3	7.4	9.4	0.2	0.2	0.3	0.2
(1,3)	3.5	6.2	7.9	8.5	0.3	0.3	0.2	0.2
(1,4)	4.2	6.1	6.9	8.9	0.2	0.3	0.2	0.3
(2,5)	2.6	4.1	5.5	9.0	0.2	0.2	0.4	0.2
(2,6)	5.8	7.0	8.5	9.6	0.3	0.3	0.2	0.2
(3,2)	1.5	2.3	3.6	4.5	0.2	0.2	0.3	0.3
(3,7)	6.5	7.2	8.3	9.4	0.5	0.2	0.2	0.1
(3,8)	5.9	7.8	8.6	9.9	0.4	0.3	0.1	0.2
(4,3)	2.1	3.2	4.5	6.8	0.2	0.2	0.3	0.3
(4,9)	1.1	2.2	3.5	4.3	0.2	0.3	0.4	0.1
(5,7)	3.2	4.8	6.7	8.2	0.2	0.2	0.3	0.3
(5,10)	6.3	7.8	8.4	9.1	0.2	0.2	0.4	0.2
(6,3)	6.8	7.7	8.5	9.6	0.4	0.1	0.1	0.4
(6,5)	0.6	1.5	3.9	5.8	0.2	0.2	0.3	0.3
(6,7)	2.1	4.8	6.6	7.5	0.2	0.4	0.2	0.2
(7,6)	4.1	6.3	8.5	9.7	0.2	0.3	0.4	0.1
(7,8)	1.6	2.8	5.2	6.0	0.2	0.3	0.3	0.2
(7,10)	1.6	3.4	8.2	9.3	0.2	0.3	0.3	0.2
(8,4)	7.0	8.0	8.8	9.4	0.2	0.2	0.2	0.4
(8,7)	2.1	4.6	8.5	9.6	0.4	0.2	0.2	0.2
(8,9)	1.7	4.9	6.5	7.8	0.2	0.2	0.2	0.4
(7,9)	3.5	4.0	5.0	7.7	0.1	0.2	0.4	0.3
(9,10)	4.6	6.4	7.6	8.9	0.4	0.1	0.2	0.3

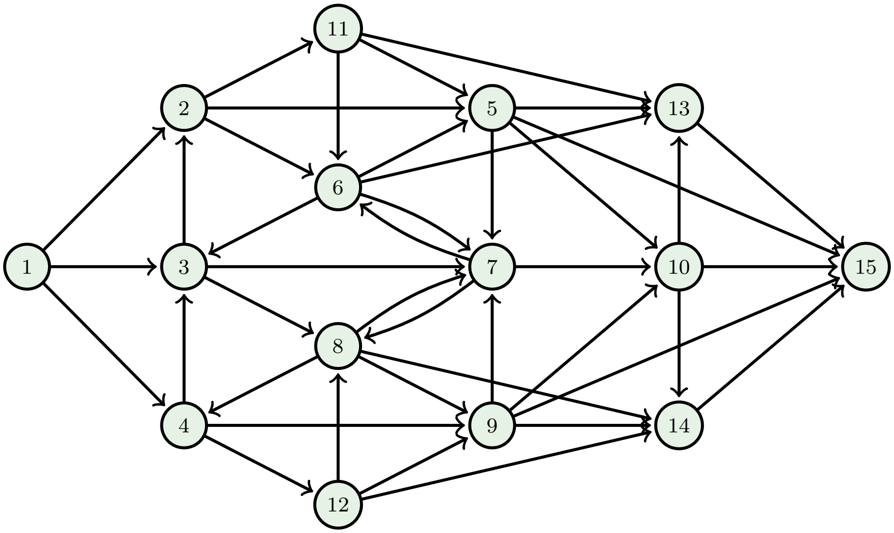

Fig. 2 Graph 4

Table 3 Weight distribution of graph 4

Edge	Weights				Probabilities			
(1,2)	16	25	36		0.6	0.3	0.1	
(1,3)	21	24	25	39	0.5	0.2	0.2	0.1
(1,4)	11	13	26		0.4	0.4	0.2	
(2,11)	24	28	31		0.5	0.3	0.2	
(2,5)	11	30			0.7	0.3		
(2,6)	13	37	39		0.6	0.2	0.2	
(3,2)	11	20	24		0.6	0.3	0.1	
(3,7)	23	30	34		0.4	0.3	0.3	
(3,8)	14	23	34		0.5	0.4	0.1	
(4,3)	22	30			0.7	0.3		
(4,9)	35	40			0.6	0.4		
(4,12)	16	25	37		0.5	0.4	0.1	
(5,13)	28	35	37	40	0.4	0.3	0.2	0.1
(5,15)	25	32			0.7	0.3		
(5,10)	27	33	40		0.4	0.3	0.3	
(5,7)	15	17	19	26	0.3	0.3	0.3	0.1
(6,5)	18	25	29		0.5	0.3	0.2	
(6,13)	21	23			0.5	0.5		
(6,7)	11	31	37		0.5	0.5	0.1	
(6,3)	18	24			0.7	0.3		
(7,10)	19	23	37		0.6	0.2	0.2	
(7,8)	12	15	22	24	0.3	0.3	0.3	0.2
(7,6)	12	23	31		0.5	0.3	0.2	
(8,7)	14	34	39		0.6	0.2	0.2	
(8,14)	14	15	27	32	0.3	0.3	0.2	0.2
(8,9)	13	31	32		0.8	0.1	0.1	
(8,4)	13	23	34		0.4	0.3	0.3	
(9,7)	10	17	20		0.6	0.3	0.1	
(9,10)	16	18	36	39	0.3	0.3	0.2	0.2
(9,15)	12	13	25	32	0.4	0.3	0.2	0.1
(9,14)	19	24	29		0.4	0.3	0.3	
(10,13)	14	20	25	32	0.3	0.3	0.2	0.2
(10,15)	15	19	25		0.4	0.3	0.3	
(10,14)	23	34			0.9	0.1		
(11,13)	13	31	25		0.6	0.3	0.1	
(11,5)	18	19	20	23	0.3	0.3	0.3	0.1
(11,6)	10	19	39		0.5	0.4	0.1	
(12,8)	15	36	39		0.5	0.3	0.2	
(12,9)	16	22			0.7	0.3		
(12,14)	10	13	18	34	0.3	0.3	0.3	0.1
(13,15)	12	31			0.9	0.1		
(14,15)	14	19	32		0.5	0.3	0.2	

Table 4 The simulation results of proposed algorithm

(a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4	
	AVI	PC	AVI	PC	AVI	PC	AVI	PC
0.0002	41,65	100	44,090	100	65,311	100	125,641	100
0.0003	28,052	100	28,947	100	36,753	100	86,982	100
0.0004	19,059	100	21,073	100	25,764	100	73,967	100
0.0005	15,393	100	16,762	100	19,652	100	43,587	100
0.0006	13,279	100	14,262	100	15,596	100	43,232	100
0.0007	11,649	100	12,410	100	12,987	100	35,500	100
0.0008	9918	100	10,557	100	10,982	100	23,340	100
0.0009	9769	100	9846	100	9718	100	31,284	100
0.0010	8402	99	8678	99	8801	100	28,440	100

(b) Average samples taken from graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4	
	TS	SPS	TS	SPS	TS	SPS	TS	SPS
0.0002	134,195	59,409	144,890	67,080	208,612	100,266	4,596,651	561,597
0.0003	89,262	38,284	94,771	43,211	117,397	59,167	4,536,043	53,7551
0.0004	60,662	27,219	68,908	31,369	82,204	41,605	241,280	147,558
0.0005	48,945	21,709	54,709	24,674	62,665	31,887	216,190	95,119
0.0006	42,051	18,079	46,409	20,566	49,632	25,283	140,702	88,108
0.0007	36,867	15,550	40,584	18,051	41,329	21,103	122,729	63,537
0.0008	34,026	13,386	34,296	15,158	34,896	17,820	77,299	46,607
0.0009	30,837	12,443	32,002	13,827	30,846	15,672	101,591	60,085
0.0010	36,753	10,739	36,323	14,928	27,939	14,239	93,121	53,188

The proposed algorithm also compared to the algorithm given in [28] and the results of comparison are given in Table 6. This table shows that the algorithm given in this paper needs smaller number of samples from the graph, but the algorithm given in [28] has higher rate of convergence. This may be due to the use of e-DLA which gathers more information.

Particle swarm optimization (PSO) is a population-based stochastic optimization technique that simulates the behaviors of social organisms in groups, such as bird and fish schooling [21]. It shares many similarities with evolutionary algorithms. PSO searches the solution space for the optimal solution using agents called particles, where trajectories of the given particles are adjusted by a stochastic and a deterministic component. Each particle is influenced by two factors: (1) the *best* achieved position of the particle and (2) the group *best* position. The particle also tends to move randomly. PSO reduces the number of parameters required for optimization algorithms, which needs lower overhead for parameter tuning. For using this strategy, PSO considers each bird (particle) in

Table 5 The simulation results of algorithm reported in [8]

(a) Average number of iterations and runs converged

a	Graph 1		Graph 2		Graph 3		Graph 4	
	AVI	PC	AVI	PC	AVI	PC	AVI	PC
0.0002	115,253	100	664,360	57	51,760	100	288,439	100
0.0003	77,129	100	430,982	59	33,030	100	275,262	100
0.0004	57,358	100	321,076	63	25,664	100	167,409	100
0.0005	46,271	100	248,827	47	20,570	100	176,932	100
0.0006	38,597	100	202,731	46	17,009	100	106,427	100
0.0007	33,218	100	175,355	46	148,54	100	103,189	100
0.0008	28,407	100	147,146	50	12,845	100	89,233	100
0.0009	25,568	100	131,513	50	11,396	100	91,521	100
0.0010	22,853	100	118,897	42	10,373	100	65,739	100

(b) Average samples taken from the graph and optimal path

a	Graph 1		Graph 2		Graph 3		Graph 4	
	TS	SPS	TS	SPS	TS	SPS	TS	SPS
0.0002	426,359	191,353	1,962,598	378,372	194,089	93,022	1,014,400	579,481
0.0003	285,136	127,104	1,271,544	248,586	124,311	60,001	927,472	596,818
0.0004	212,419	95,123	935,895	187,074	96,236	46,158	577,348	339,401
0.0005	171,089	76,453	692,395	132,649	77,164	36,976	592,380	381,695
0.0006	142,766	63,687	564,279	108,945	63,777	30,730	369,042	218,752
0.0007	122,827	54,792	464,273	89,844	55,610	26,721	352,776	214,076
0.0008	105,241	47,077	409,822	81,742	48,205	23,115	305,305	185,616
0.0009	94,604	42,265	357,008	71,284	42,742	20,540	308,579	194,580
0.0010	84,583	37,789	312,065	60,180	38,809	18,707	227,192	135,282

Table 6 Comparison of the proposed algorithm and the algorithm given in [28] for graph 4

a	The proposed algorithm			The algorithm given in [28]		
	AVI	PC	TS	AVI	PC	TS
0.01	1188	100	3819	1391	100	5968
0.02	778	100	2316	747	100	3193
0.03	601	100	1656	531	100	2250
0.04	375	98	947	415	100	1755
0.05	319	97	729	319	100	1354
0.06	286	95	524	316	98	1339
0.07	244	95	456	222	100	953
0.08	174	93	279	257	100	1060
0.09	153	92	242	267	96	1098

Table 7 The simulation results of PSO-based algorithm reported in [33]

Swarm size	PC of graph 1	PC of graph 2	PC of graph 3	PC of graph 4
25	67.33	33.33	33.33	33.33
50	100	91.66	66.66	66.66
100	91.66	75	75	50

Table 8 The simulation results of ant colony algorithm (number of ants = 10, pheromone concentration rate = 0.9, evaporation rate = 0.6), Q-learning (step size = 0.7, discount rate = 0.9), and actor–critic learning algorithm (step size = 0.7, discount rate = 0.9)

(a) Average number of iterations and runs converged

Algorithm	Graph 2		Graph 3		Graph 4	
	AVI	PC	AVI	PC	AVI	PC
Ant colony	17,300	85	25,400	75	36,045	70
Q-learning	21,500	84	37,752	77	40,805	67
Actor–critic	19,742	87	33,128	74	38,562	71

(b) Average samples taken from graph and optimal path

Algorithm	Graph 2		Graph 3		Graph 4	
	TS	SPS	TS	SPS	TS	SPS
Ant colony	404,760	156,730	434,272	213,579	1,569,742	5,173,714
Q-learning	61,217	27,469	78,210	33,134	91,351	42,196
Actor–critic	57,641	23,827	67,415	31,372	82,684	36,734

the search space as a single solution. The proposed algorithm also compared with PSO-based algorithm proposed in [33]. These algorithms are run 100 times on the four test graphs described before, and the results are given in Table 7. This table reports the percentage of runs converged with respect to the swarm size. The results shown in this table shows that algorithm 1 has a higher rate of convergence for all graphs.

In this paper, we also designed some algorithms based on ant colony [15], Q-learning [50], and actor–critic [50] for finding the shortest path. These algorithms are also tested 100 times on the four test graphs, and the results are given in Table 8. We tried to find the best parameters for the all three mentioned algorithms by trial and error and the best results are reported in these tables. The AVI for the algorithm based on ant colony based is the average number of iterations that all ants traversed on the path. This table shows that the proposed algorithm requires a fewer number of iterations and also requiters a fewer number samples from graph edges. The algorithm given in this paper and the algorithm based on ant colony are distributed, while algorithms based on Q-learning and actor–critic learning algorithms are centralized.

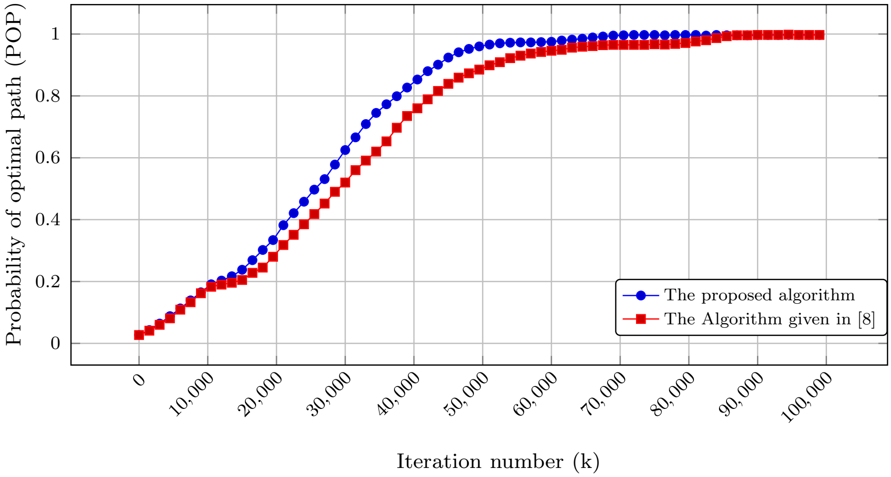

Fig. 3 Probability of optimal path for graph 1

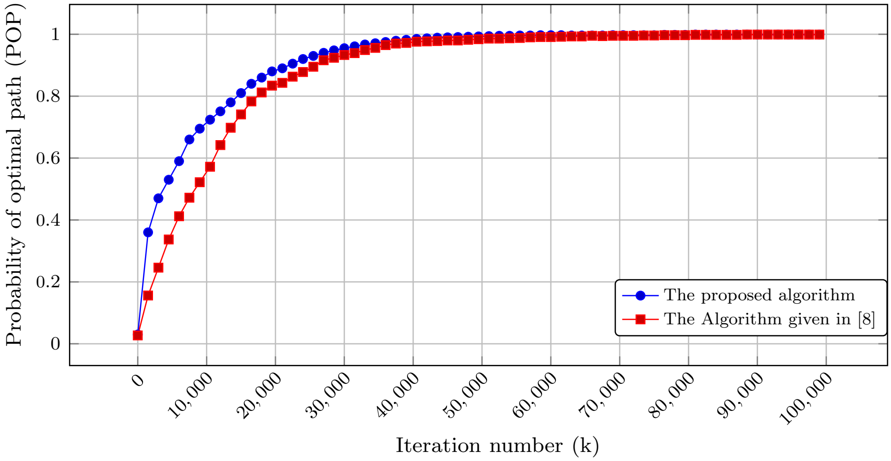

Fig. 4 Probability of optimal path for graph 2

Figures 3, 4, 5, and 6 show the probability of selecting the shortest path for different graphs for a typical run as algorithm proceeds. These figures show that POP increases as time increases and approaches to the unity as time goes on. These figures also show that the algorithm given in this paper finds the shortest path in a smaller number of iterations. A useful property of iterative algorithms is its *any time behavior*. This property shows that when the execution of an iterative algorithm stopped, what we can say about the quality of the generated solution. We study this property by plotting the change in the POP versus time (see Figs. 3, 4, 5 and 6). These curves show a rapid increase of POP and then the increase will become flattening, which shows that we do not need long run-time

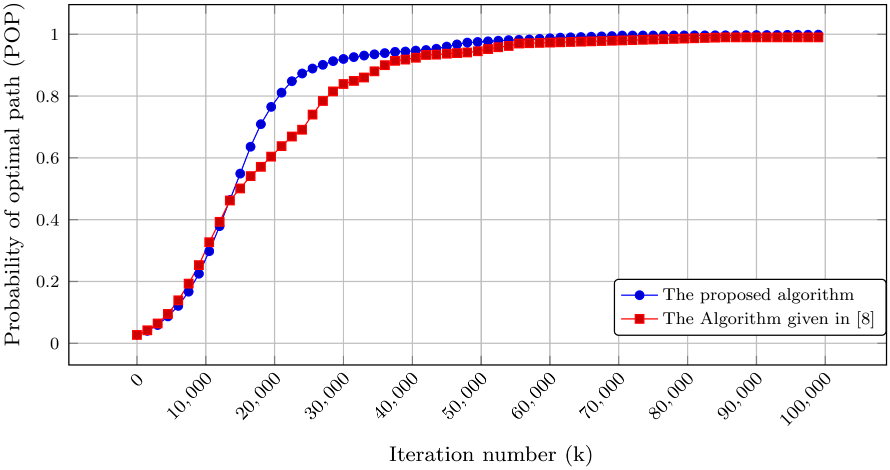

Fig. 5 Probability of optimal path for graph 3

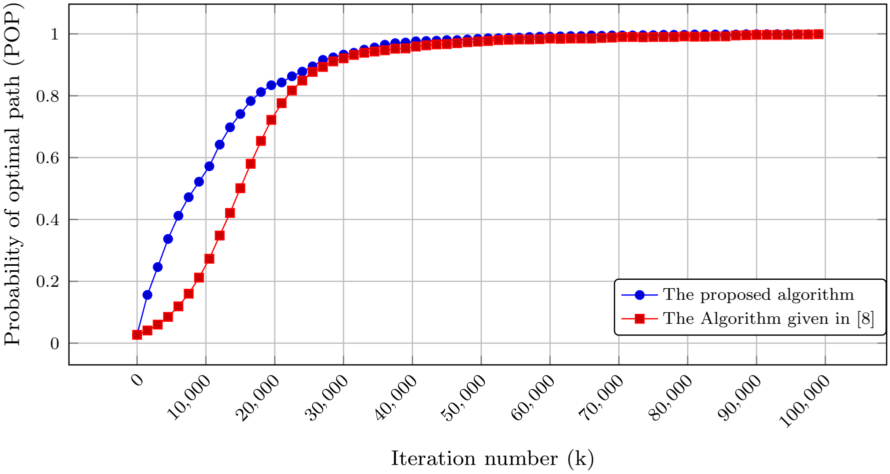

Fig. 6 Probability of optimal path for graph 4

of the algorithm for producing a suboptimal solution. This property holds for many greedy algorithms starting from a solution and then improving the quality of the solution, which is an advantage over constructive greedy algorithms. Its name shows that when we stop the improvement at any iteration, it will produce a solution, even it is suboptimal. We can use this curve for finding an appropriate termination condition. Any point (x, y) in anytime curve shows that if we fix the probability of finding the shortest path to y, then in average we require x iterations for finding the shortest path. Figures 3, 4, 5 and 6 show that for a fixed POP, the algorithm given in this paper finds the shortest path in smaller number of iterations than the algorithm reported in [8].

3.1 Analysis of the results

The results of experiments show that in most cases the proposed algorithm requires fewer iterations and hence fewer samples of graph edges to find the shortest path in the stochastic graph. Hence, in most cases the proposed algorithm outperforms the related algorithm. In what follows, we explain some points regarding the behavior of the proposed algorithm, which are concluded from the experimental results.

1. The experimental results show that the percentage of runs that correctly finds the shortest path increases as the learning rate decreases. As an example considering graph 2, 100% of runs converge when $a = 0.0002$, whereas it decreases to 99% when $a = 0.001$ (Table 4). The reason for this observation is that when the learning rate decreases the action probability vectors are updated with smaller values, and hence, action probability vectors are updated smoothly. This type of updating decreases the probability of finding wrong paths. Hence, the percentage of runs that correctly finds the shortest path increases. Since by decreasing the learning rate, the action probability vectors are updated with a smaller value; then, it increases the number of iterations, and hence, the number of samples required to find the shortest path is also increased.
2. The results of experiments show that the percentage of runs that correctly finds the shortest path for the algorithm given in this paper is higher than the percentage of runs that correctly finds the shortest path for the PSO-based algorithm given in [33].
3. In the proposed algorithm, according to the scheme used to update learning rates, as approaching the destination node, learning rate, a, for automata gets larger. This updating scheme for learning rates causes automata closer to the destination node converge more quickly. Faster convergence of automata closer to the destination node increases the slop of "any time" curve and results in reaching a pre-specified threshold P_{pop} with fewer iterations.
4. The results of experiments show that the number of iterations for finding the shortest path increases as the difference between the weight of the shortest and the weight of the second best shortest path decreases. In fact, as the number of paths with weights closer to the weight of the shortest path increases, the number of iterations required for finding the shortest path also increases. This means discriminating between the shortest paths from other paths becomes more difficult. This is same as having an environment in which we have several actions with almost the same action penalty probabilities. Finding the optimal action in such environment needs too many interactions between the automaton and the environment. Hence, in the stochastic shortest path problem, this means we need more samples from the edges and consequently more iteration, and hence, the running time of the algorithm will be increased. In most cases, the algorithm given in this paper requires a fewer number of samples in comparison with the algorithm given in [8]. Since the actual running time of algorithm is proportional to the number of required edges samples, we may conclude that the algorithm given in this paper needs a smaller running time in comparison with the algorithm given in [8].

5. The running time of the algorithm can be computed from the total number of samples taken from the edges of the graph. For such a sampling of an edge, we have two computations: one for selection of an edge (corresponding to the selection of an action) and updating the action probability vector of the given automata. These running times are in order of $O(r)$, where r is the number of actions of the automata. The order of these running times is same for almost all learning automata algorithms. Hence, the algorithm requiring a fewer number of samples runs faster. From the experimental results, it is evident that the proposed algorithms find the shortest path in a smaller time for most cases.
6. The running time and the percentage of runs converged of the proposed algorithm in addition to the length of the shortest path and the second shortest path mentioned above depend on two other factors: the out degrees of nodes and the number of nodes in the shortest path. When the degree of nodes is increased, the initial probability of selecting actions become smaller and action probabilities are updated with smaller steps. Hence, the running time of the algorithm becomes larger. In the other hand, increasing the out degrees of node increases the number of paths from the source node to the destination node. This decreases the percentage of runs converged, because the probability of finding the shortest path decreases. When the number of nodes in the shortest path increases, *pop* started from a smaller value and updated with smaller steps. Hence, increasing the number of nodes in the shortest path increases the running time of the algorithm. This also increases the probability of finding a path other than the shortest path. Hence, increasing the number of nodes in the shortest path decreases the percentage of runs converged. Thus, it can be concluded that the number of times actions selected by LAs of DLA, which also equals to the number of edges samples, is very much dependent on the input graph.

4 Conclusions

We have considered the problem of finding the shortest path in stochastic graphs. The algorithm given in this paper provides a procedure for finding a path with minimal expected weight from the source node to the destination node with minimal expected weight. The given algorithm is an iterative procedure by using a DLA. The comparison between the algorithm given in this paper and some other related algorithms based on PSO, ant colony, Q-learning, and actor–critic learning algorithms, show that the percentage of runs that finds the shortest path correctly for these algorithms is lower than the percentage of runs that finds the shortest path correctly for the algorithm given in this paper and the number of iterations needed by these algorithms are higher than the average iteration needed by the algorithm given in this paper. Also comparison with other DLA-based algorithms shows that the number of edge needed to be sampled is smaller than other algorithms, but it has also a smaller rate of convergence than the related algorithms. The reason for these results is the definition of dynamic thresholds. In reinforcement learning, the number of interaction between the agent and the environment is very important and the goal is to decrease the number of interactions as possible. It has been also shown that when all

LAs are using the L_{R-I} algorithm, it finds the shortest path with a high probability. This probability can be made close to unity as desired. The main steps of proof were given, and the detail of proof is dependent to the underlying structure of the graph. The results obtained from experiments show that the algorithm given in this paper needs fewer samples from edges.

For future works, we are planning to have a proof that can be used for every graph.

Acknowledgements The authors would like to thank anonymous reviewers for their time, valuable comments, constructive criticism, and suggestions which greatly improved the paper.

References

1. Adlakha VG (1986) An improved conditional Monte Carlo technique for the stochastic shortest route problem. Manag Sci 32(10):1360–1367
2. Alexopoulos C (1997) State Space partitioning methods for stochastic shortest path problems. Networks 30:9–21
3. Beigy H, Meybodi MR (2000) Adaptation of Parameters of BP algorithm using learning automata. In: Proceedings of VI Brazilian Symposium on Neural Networks, SBRN2000, Brazil, pp 24–31
4. Beigy H, Meybodi MR (2000) Solving the graph isomorphism problem using learning automata. In: Proceedings of 5th Annual International Computer Society of Iran Computer Conference, CISCC-2000, Tehran, Iran, pp 402–415
5. Beigy H, Meybodi MR (2001) Backpropagation algorithm adaptation parameters using learning automata. Int J Neural Syst 11(3):219–228
6. Beigy H, Meybodi MR (2002) Call admission control in cellular mobile networks: a learning automata approach, vol 2510. Springer-Verlag lecture notes in computer science. Springer, Berlin, pp 450–457
7. Beigy H, Meybodi MR (2005) An adaptive call admission algorithm for cellular networks. Electr Comput Eng 31(2):132–151
8. Beigy H, Meybodi MR (2006) Utilizing distributed learning automata to solve stochastic shortest path problems. Int J Uncertain Fuzziness Knowl Based Syst 14(5):591–615
9. Beigy H, Meybodi MR (2009) Adaptive limited fractional guard channel algorithms: a learning automata approach. Int J Uncertain Fuzziness Knowl Based Syst 17(6):881–913
10. Beigy H, Meybodi MR (2009) Cellular learning automata based dynamic channel assignment algorithms. Int J Comput Intell Appl 8(3):287–314
11. Beigy H, Meybodi MR (2009) A learning automata-based algorithm for determination of the number of hidden units for three layer neural networks. Int J Syst Sci 40(1):101–118
12. Chen BY, Shi C, Zhang J, Lam WHK, Li Q, Xiang S (2017) Most reliable path-finding algorithm for maximizing on-time arrival probability. Transp B Transp Dyn 5(3):1–17
13. Das A, Martel C (2009) Stochastic shortest path with unlimited hopes. Inf Process Lett 109:290–295
14. Deo N, Pang C (1984) Shortest path algorithms: taxonomy and annotation. Networks 14:275–303
15. Engelbrecht AP (2007) Computational intelligence: an introduction. Wiley, Hoboken
16. Fan YY, Kalaba RE, Moore JE (2005) Shortest paths stochastic networks with correlated link costs. Comput Math Appl 49:1549–1564
17. Frank H (1969) Shortest path in probabilistic graphs. Oper Res 15:583–599
18. Fu L, Rilett LR (1998) Expected shortest paths in dynamic and stochastic traffic networks. Transp Res Part B Methodol 32:499–516
19. Guillot M, Stauffer G (2019) The stochastic shortest path problem: a polyhedral combinatorics perspective. Eur J Oper Res (**in press**)
20. Hasanzadeh M, Meybodi MR (2014) Grid resource discovery based on distributed learning automata. Computing 96(9):909–922
21. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of International Conference on Neural Networks (ICNN'95), pp 1942–1948

22. Lakshmivarahan S, Thathachar MAL (1976) Bounds on the convergence probabilities of learning automata. IEEE Trans Syst Man Cybern SMC–6:756–763
23. Martin JJ (1965) Distribution of the time through a directed acyclic network. Oper Res 13:46–56
24. Meybodi MR, Beigy H (2001) Neural network engineering using learning automata: determining of desired size of three layer feedforward neural networks. J Fac Eng 34(4):1–26
25. Meybodi MR, Beigy H (2002) A note on learning automata based schemes for adaptation of BP parameters. J Neurocomput 48(3):957–974
26. Meybodi MR, Beigy H (2002) New learning automata based algorithms for adaptation of backpropagation algorithm parameters. Int J Neural Syst 12(1):45–68
27. Meybodi MR, Laksmivarahan S (1983) A learning approach to priority assignment in a two class M/M/1 queueing with unknown parameters. In: Proceedings of the Third Yale Workshop on Applications of Adaptive Systems Theory, pp 106–109
28. Meybodi MRM, Meybodi MR (2014) Extended distributed learning automata: an automata-based framework for solving stochastic graph optimization problems. Appl Intell 41(3):923–940
29. Miller-Hooks E (2001) Adaptive least-expected time paths in stochastic, time-varying transportation and data networks. Networks 37(1):35–52
30. Miller-Hooks E, Mahmassani H (2000) Least expected time paths in stochastic, time-varying transportation networks. Transp Sci 34:198–215
31. Mirchandani PB (1970) Shortest distance and reliability of probabilistic networks. Comput Oper Res 8:347–355
32. Mirchandani PB, Soroush H (1985) Shortest distance and reliability of probabilistic networks: a case with temporary preferences. Comput Oper Res 13(4):365-087
33. Momtazi S, Kafi S, Beigy H (2008) Solving stochastic path problem: particle swarm optimization approach, vol 5027. Springer-Verlag lecture notes in artificial intelligence, Springer, Berlin, pp 590–600
34. Moradabadi B, Beigy H (2014) A new real-coded Bayesian optimization algorithm based on a team of learning automata for continuous optimization. Gen Program Evol Mach 15(2):169–193
35. Moradabadi B, Meybodi MR (2017) Link prediction in fuzzy social networks using distributed learning automata. Appl Intell 47(3):837–849
36. Moradabadi B, Meybodi MR (2017) Link prediction in stochastic social networks: learning automata approach. J Comput Sci 127:313–328
37. Narendra KS, Thathachar KS (1989) Learning automata: an introduction. Printice-Hall, New York
38. Nedzelnitsky OV, Narendra KS (1987) Nonstationary models of learning automata routing in data communication networks. IEEE Trans Syst Man Cybern SMC–17(6):1004–1015
39. Oommen BJ, Roberts TD (2000) Continuous learning automata solutions to the capacity assignment problem. IEEE Trans Comput 49(6):608–620
40. Oommen BJ, de St. Croix EV (1996) Graph partitioning using learning automata. IEEE Trans Comput 45(2):195–208
41. Peer SK, Sharma DK (2007) Finding the shortest path in stochastic networks. Comput Math Appl 53:729–740
42. Pineda LE, Wray KH, Zilberstein S (2017) Fast SSP solvers using short-sighted labeling. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4–9, 2017, San Francisco, California, USA, pp 3629–3635
43. Polychronopoulos GH, Tsitsiklis JN (1990) Stochastic shortest path problems with recourse. Networks 27:133–443
44. Pritsker AAB (1966) Application of multi-channel queuing results to the analysis of conveyor systems. J Ind Eng 17:14–42
45. Rezvanian A, Meybodi MR (2015) Finding maximum clique in stochastic graphs using distributed learning automata. Int J Uncertain Fuzziness Knowl Based Syst 23(1):1–32
46. Saghiri AM, Khomami MD, Meybodi MR (2019) Intelligent random walk: an approach based on learning automata. Springer, Berlin
47. Sigal CC, Pritsker AAB, Solberg JJ (1980) The stochastic shortest route problem. Oper Res 48:1122–1129
48. Sizal CU, Pritsker VAQ, Solberg JJ (1979) The use of cutsets in Monte-Carlo analysis of stochastic networks. Math Comput Simul 21:276–384
49. Srikantakumar PR, Narendra KS (1982) A learning model for routing in telephone networks. SIAM J Control Optim 20(1):34–57
50. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge

51. Teichteil-Königsbuch F (2012) Stochastic safest and shortest path problems. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, pp 1825–1831
52. Thathachar MAL, Ramakrishnan KR (1981) A hierarchical system of learning automata. IEEE Trans Syst Man Cybern SMC–11(3):236–248
53. Thomas B, White C (2007) The dynamic shortest path problem with anticipation. Eur J Oper Res 176(2):836–854
54. Vahidipour S, Esnaashari M, Rezvanian A, Meybodi M (2019) Gapn-la: a framework for solving graph problems using petri nets and learning automata. Eng Appl Artif Intell 77:255–267
55. Waller ST, Ziliaskopoulos AK (2002) On the online stochastic shortest paths with limited arc cost dependencies. Networks 40(4):216–227
56. Zeng W, Miwa T, Wakita Y, Morikawa T (2015) Application of lagrangian relaxation approach to α-reliable path finding in stochastic networks with correlated link travel times. Transp Res Part C Emerg 56(3):309–334