A sampling method based on distributed learning automata for solving stochastic shortest path problem

Hamid Beigy[a,*], Mohammad Reza Meybodi[b]

[a]*Sharif Intelligent Systems Laboratory, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
[b]*Soft Computing Laboratory, Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran*

Abstract

This paper studies an iterative stochastic algorithm for solving the stochastic shortest path problem. This algorithm by using a distributed learning automata tries to find the shortest path by taking a sufficient number of samples from the edges of the graph. In this algorithm, the edges to be sampled are determined dynamically as the algorithm proceeds. At each iteration of this algorithm, a distributed learning automata used to determine the edges to be sampled. This sampling method using distributed learning automata reduces the number of samplings from edges which may not be along the shortest path resulting in a reduction in the number of edges to be sampled. In this paper, we propose a new method for analysis of the algorithm. The method proposed in this paper, unlike the previous ones which were based on Martingale theory, is based on the sampling. The proof given in this paper, unlike the previous ones which were for a specific input graph, is not restricted to the input graph and is general. We also show that as the number of samples taken from the edges increases, the probability of finding the shortest path increases. Experimental results obtained from some benchmark stochastic graphs and some random graphs also confirm the theoretical results.

Keywords: Distributed learning automata, learning automata, shortest path problem, stochastic graph

1. Introduction

Many problems can be modeled as finding the shortest path in weighted graphs. In many applications, such as travel time in intelligent transportation systems, length of edges are random variables and these graphs are called *stochastic graphs*. A stochastic graph G can be defined by a triple $G = (V, E, \mathcal{F})$ comprising a set V of nodes with a set E of edges which are ordered pairs of elements of nodes, and the probability distribution $\mathcal{F}$ associated to edges of the graph describing the statistics of edge lengths. In this graph, the length of edge (i, j) is sampled from distribution defined by f_{ij}. Node j is successor of node i and node i is predecessor of node j. We also assume that the length of edge (i, j) is a positive-valued random variable with f_{ij} as its probability

*Corresponding author
Email addresses: beigy@sharif.edu (Hamid Beigy), mmeybodi@aut.ac.ir (Mohammad Reza Meybodi)

density function. It is also assumed that these probability density functions are unknown the algorithm, which only has access to samples taken from the corresponding distribution. For example, considering transportation networks, the intersections represent nodes of the graph and each edge represents the street/road between the two corresponding nodes. In this network, measures such as the travel time, the distance, and the cost of a traveling show the edge length. The travel time is a random variable with unknown distribution. The travel time in each street/road depends on the traffic situation on that edge (street/road). When traveling from each street/road, we have a sample travel time, which is sampled from the corresponding distribution. This problem in *intelligent transportation systems* is called vehicle *route guidance systems (RGS)*. The goal of solving the RGS problem is the design of an algorithm for finding the optimal route (path) from the origin to the destination. This optimal route between an origin and destination for most RGS is defined as the one with minimum expected travel time [1].

A sequence $\left(v_{i_1}, v_{i_2}, \ldots, v_{i_{n_i}}\right)$ of distinct nodes of the graph with the property that there is an edge between every two consecutive pair of nodes, $(v_{i_j}, v_{i_{j+1}}) \in E$ (for $1 \leq j < n_i$), is called *simple path*, where n_i is the number of nodes in the given path. Since each edge length is a random variable, the goal is to find its expected value. Let C_{uv} denotes a sample length of edge (u, v) taken form the probability density function f_{uv} and let $\bar{C}_{uv}$ denotes the expected length of the edge (u, v). We denote a simple path from v_s to v_d as π and its expected length as $\bar{L}_\pi$. The expected length of path π denoted by $\bar{L}_\pi$ equals to $\sum_{j=1}^{n_i-1} \bar{C}_{i_j i_{j+1}}$. We also assume that the graph has t distinct simple paths $\Pi = \{\pi_1, \pi_2, \ldots, \pi_t\}$ from v_s to v_d. A simple path with minimum expected length from v_s and to v_d is called the *shortest path* and denoted by π^*, i.e. the shortest path π^* has expected length of $\bar{L}_{\pi^*} = \min_{\pi \in \Pi}\{\bar{L}_\pi\}$. This means that path π^* has the smallest expected length among all paths from the source node v_s to the destination node v_d. The problem of finding the shortest path in a stochastic graph called *stochastic shortest path problem* (SSPP). A simple algorithm for solving this problem is to take a sufficient number of samples from edge length C_{uv} from each edge (u, v) and then estimate the expected length $\bar{C}_{uv}$ of that edge. Finally, by using an algorithm for solving shorted path problem, such as Dijkstra or Floyd-Warshal, we can find the shortest path from the source node v_s to the destination node v_d. This approach has two main problems: the first one is that we need a lot of samples from each node for estimating the expected length with high confidence and the second one is that we need a lot of samples from the edges that don't belong to the shortest path. The goal of the approach proposed in this paper is to take more samples from edges belonging to the shortest path and take a smaller number of samples from the edges that don't belong to the shortest path.

Several algorithms are proposed in the literature for solving the SSPP. These algorithms can be classified into three groups based on the time that values of edge lengths are learned [2]. (1) The lengths of edges are

learned before traversing a path [3]. (2) The length of an edge is never learned or become known after traversing the shortest path [4]. (3) The lengths of edges are learned as traversing different paths in the graph.

Let π be a path and the probability that this path is shortest be denoted by $\mathbb{P}[\pi]$. An approach for finding $\mathbb{P}[\pi]$ is to use distributed learning automata (DLA), in which a network of learning automata cooperate for solving the stochastic shortest path problem. Several DLA-based algorithms are proposed in the literature for solving the SSPP. These algorithms fall in the third group in which the lengths of edges are learned as traversing different paths in the graph. Beigy and Meybodi proposed some DLA-based algorithms for solving SSPP [5, 6]. The convergence of these algorithms was shown by using Martingale Theorem. Meybodi and Meybodi proposed an algorithm based on an extended version of DLA for solving the SSPP and its convergence was shown by using Martingale Theorem [7]. Vahidipour et al. proposed an algorithm for solving the SSPP by using a combination of LAs and Pertri-nets [8]. This algorithm explores different paths in parallel by sending different tokens. The convergence of this algorithm was shown by using a continuous-time Markov chain. The combination of LA and Petri-nets was generalized for solving different graph problems [9].

In this paper, an iterative algorithm using DLA for solving the SSPP, which falls in the third group, is studied. This algorithm tries to find the shortest path, π^*, from the set of paths $\Pi = \{\pi_1, \ldots, \pi_t\}$ by taking a sufficient number of samples from the lengths of their edges. This algorithm dynamically determines the next edge to be sampled. This method reduces the number of unnecessary sampling from the edges that are not along the shortest path resulting reduction of the overall number of samples. The algorithm studied in this paper is obtained by a modification on the algorithm given in [6], by changing the updating rule of learning rates but the different method was used for proof of convergence. The convergence of the algorithms proposed in [5, 6] was shown by using Martingale theory, and the outline of proof is given and then the convergence of the algorithms was proved only for a five nodes graph. When the size of the graph becomes larger, the derivation of equations using the methods given in [5, 6] becomes very complicated or even intractable. Hence, the convergence of algorithms are not applicable for general graphs and must be done for every input graph. In this paper, a new method of proof based on the sampling, which is different from [5, 6], is used to prove that if each LA of DLA uses the linear reward-inaction learning (L_{R-I}) algorithm, the algorithm finds the shortest path with a high probability. The L_{R-I} algorithm only updates its state when the environment gives a reward to the selected action. The proof shows that if the parameters of the algorithm are chosen properly, it converges to the path with the minimum expected length with probability as close to unity as desired. Hence, the convergence proof given for this algorithm in contrast to the previously reported convergence method for DLA-based algorithms is based on sampling and can be used for the arbitrary graph. We also evaluate the modified method using

some benchmark stochastic graphs, some randomly generated graphs, and Kronecker graphs. The experimental results have shown that the modified algorithm outperforms the algorithms reported in [5, 6]. Specifically, the algorithm given in this paper needs a smaller number of iterations and a smaller number of samples from the edges of the graph than the algorithm given in [5, 7] to find the shortest path at approximately the same convergence rate. But the modified algorithm increases the number of iterations and the number of samples from the edges of the graph than the algorithm given in [6] to find the shortest path by increasing the rate of convergence.

The rest of this paper is organized as follows: The related work is given in section 2. LA and DLA are briefly described in section 3. The modified algorithm for solving the stochastic shortest path problem is given in section 4. In this section, we also study the convergence of this algorithm and propose the new method for proving the convergence of the algorithm. In section 5, the experimental results are reported. In section 6, we study the computational complexity of the modified algorithm and compare it with the computational cost of three other related algorithms. Finally, section 7 concludes the paper.

2. Related work

In this section, we give a brief review of the related work on algorithms for solving the stochastic shortest path problem. The SSPPs can be categorized into two main groups: the first group aims to find a priori solution that minimizes the expected length and the second group computes an on-line solution. Some problems for general stochastic graphs have been analyzed in [3, 10]. The unique arcs concept was studied in [11], and the uniformly directed cuts were used by Sigal et al. [12] for the analysis of the SSPPs. In these research-works, the required probabilistic quantities are represented by multiple integrals and numerical calculation of such integrals quickly gets out of hand, even for small networks [13]. Since the exact computation of the distribution of path lengths is difficult, Monte Carlo simulation was used [14]. Some researchers used state-space partitioning to solve the SSPP [15]. Two models with recourse are reported in [2]. The first one assumes general spatial dependence among the edges and the second one assumes that lengths are IID random variables. Then two exponential-time algorithms are proposed. The SSPP with recourse on time-varying networks studied in [16, 17], and two algorithms proposed. In [18], the SSPP with limited inter-edge dependency is considered. These research works assumed that distributions for edges lengths are known. In [19], a dynamic programming approach was used to find the shortest path when the number of edges of each path is limited. In [20], it assumed that the length for edges in the graph are exponential distributed random variables and finding the shortest path with minimal expected length formulated as a linear programming problem. In [21], a network setting used in which the

length of each edge is a random variable with the known probability distribution. In [22], it has shown the SSPP with the limited number of hops, i.e., paths with at most $n-1$ edges is an NP-Hard problem, where n is the number of nodes in the graph. Then they extended the results of [19] for the SSPP with an unlimited number of hops. In [23], 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. [24]. Guillot and Stauffer proposed a new framework for solving SSPP infinite state and action spaces in the context of MDP [25].

The stochastic shortest path with correlated link travel times is considered in [26]. It is assumed that edge and path lengths are respectively fitted by the log-normal and normal distribution. The authors of [27] proposed algorithms to find the most reliable path in large-scale road networks. It is assumed that path length follows normal distributions and link lengths are mutually independent.

In [5, 6], two distributed learning automata (DLA) based algorithms for solving the SSPP are reported. In the first algorithm, a DLA constructed from the graph and learning automata (LA) in the DLA update their action probability vectors until the shortest path has been found. The second algorithm is obtained from the first algorithm by modifying the definition of the dynamic threshold. This modified algorithm requires a smaller number of samples to find the shortest path. Misra and Oomenn considered dynamic all-pairs shortest–path routing problem using a fast–converging pursuit automata learning approach [28]. In [7], an extended version of DLA was proposed in which each LA has an activity level and only the LA with the highest activity level can perform an action on the environment. The experimental results reported in [7] shows that this algorithm requires new samples from the graph than the algorithm given in [5]. Vahidipour et al proposed an algorithm based on the combination of LA and Petri nets to speedup the process of finding the stochastic shortest path [8]. In this algorithm, paths are traversed in parallel by sending token on the all outgoing-edges of every node. Then, the estimated time to visit a node is measured and using this estimated time, the shortest path is found. Vahidipour et al. proposed a framework that generalizes different frameworks for the fusion of LAs and Petri nets for solving different graph-based problems [9]. The above-mentioned algorithms are deferent from each other from the respect to (1) the learning algorithm of LAs, (2) the updating algorithm for learning rate, (3) the activation strategy of LAs, (4) the method for avoiding cycles, and (4) the direction in which LAs update their action probability vectors.

3. Distributed learning automata

The learning in learning automata (LA), which is an interconnection of a stochastic automaton and a random environment, is to find the best action from its action set. At the stage k, the automaton selects its action from the finite set of r actions, denoted by $\underline{\alpha} = (\alpha_1, \ldots, \alpha_r)$, using the action probability vector $\underline{p}(k) = (p_1(k), \ldots, p_r(k))$, where $p_i(k)$ is the probability of selecting action α_i at stage k. Let this action denoted by $\alpha(k) = \alpha_i$, which applied to the random environment. The random environment emits a reinforcement signal $\beta(k) \in \{0, 1\}$, which is a stochastic signal. The environment penalizes the selected action $\alpha(k) = \alpha_i$ with probability c_i by producing the reinforcement signal $\beta(k) = 1$ and rewards that action with the reward probability of $d_i = 1 - c_i$ by producing the reinforcement signal $\beta(k) = 0$. Finally, the automaton updates its action probability vector using the received reinforcement signal by employing a learning algorithm. Note that the set of penalty probabilities $\underline{C} = \{c_1, \ldots, c_r\}$ is unknown to the LA initially and its goal is to find the action for which the penalty probability is minimum as the automaton interacts with the environment.

The L_{R-I} learning algorithm, which is used later in this paper, updates the probability of choosing action α_j (for all $j = 1, 2, \ldots, r$) using the following rule when at stage k, the learning automaton chooses action α_i.

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

when $\beta(k) = 0$ and $\underline{p}(k)$ remains unchanged when $\beta(k) = 1$. Parameter $a_k \in (0, 1)$ is *learning rate* at stage k and determines the amount of increases (decreases) of the action probabilities.

LA widely used in applications such as telephone and data network routing [29], solving NP-Complete problems [30, 5], capacity assignment [31], neural networks engineering [32, 33], social networks [34] design of Bayesian optimization algorithms [35], cellular networks [36, 37], classification and clustering [38], intelligent random walks [39] to mention a few.

A single LA has a limited capability; while by using collectively cooperating LAs, more capabilities will be obtained. Distributed learning automata (DLA) is a graph of automata in which LAs are cooperating for solving a problem. In this graph, the set of nodes shows the set of LAs and the set of outgoing edges of any node shows the action set of the corresponding LA. A sample DLA with three LAs, each of which has two actions, is shown in Figure 1. In this DLA, when A_1 chooses action α_1, then A_2 is activated, which is in the other end of the corresponding edge. After that A_2 chooses one of its actions. The selected action of A_2 also activates another LA connected to A_2.

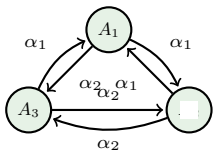

Figure 1: Distributed Learning Automata

Formally, a DLA can be defined by a graph (A, E) comprising a set A of LAs with a set $E \subset A \times A$ of edges. Let $A_i \in A$ be the ith automaton which has r_i actions. Let also $\underline{\alpha}_i = \left(\alpha_1^i, \ldots, \alpha_{r_i}^i\right)$ be the action set of automaton A_i and $\underline{p}^i(k) = \left(p_1^i(k), \ldots, p_{r_i}^i(k)\right)$ be its action probability vector at stage k, where $p_j^i(k)$ denotes the probability of selecting action α_j^i by A_i (i.e. action α_j of LA A_i) at the stage k. We use the superscript for α and $\underline{p}$ to denote the index of an automaton and the subscript to denote the index of an action. An action of a DLA is a sequence of actions for its constituting LAs. For example, for DLA given in Figure 1, one action is $\left(\alpha_1^1, \alpha_2^2, \alpha_1^3\right)$, which shows tour (v_1, v_2, v_3, v_1) in the corresponding graph

Learning in the DLA can be described as follows. The DLA chooses an action and applies it to the random environment. The environment generates a reinforcement signal, which will be used by DLA to update its action probability vectors according to a learning algorithm. The DLA chooses its action in the following manner. At the beginning of any round of action selection, only one LA is active. The activation of this LA is application dependent. For example in the SSPP, the source LA is activated first. The activated LA chooses one of its actions. The chosen action will activate another LA on the other end of the edge corresponding to the chosen action. The procedure of activating a LA and passing the activation to the next LA continues until no LA will be activated as a result of the action selection. For example, to find tours for DLA given in Figure 1, LAs sequentially choose their actions until the first activated LA is reached or $n = 3$ actions are chosen, where n is the number of LAs in the DLA.

When the DLA is being used in any application, the following items must be specified: (1) the underlying graph structure of DLA, (2) choosing LAs of DLA including their action sets and learning algorithms, (3) the action selection mechanism of DLA, and (4) the updating mechanisms for updating action probability vectors of LAs in the DLA. When the DLA is used for solving graph problems as the shortest path problem, the actions of DLA specify different paths from the source node to the destination node of the graph. Hence, we considered that the graph of DLA is isomorphic to the input graph. In this problem, LAs of DLA correspond to the nodes of the graph and each action of an LA corresponds to an outgoing edge of the corresponding node. The learning algorithm specifies how to update the probability of choosing different paths. However, the graph can be summarized and each hyper-node can be an LA. For example, we can find the strongly connected components

of the graph and assign an LA to each component. Then, the problem is solved for each hypernode recursively. We don't use this approach, because of the two following reasons. The first reason is that for summarizing the graph, we need to sample different edges, which is a costly process. The second reason is that the convergence to the shortest path can't be guaranteed. When the input graphs are deterministic or sufficient statistics of edges' lengths are available, this summarization approach helps to improve the running time of the algorithm, but the convergence to the optimal solution depends on the graph summarization and also the algorithm for solving the problem.

DLA have been applied successfully to many applications such as solving the SSPP [5, 7], grid resource discovery [40], finding Maximum clique [41], and link prediction in social networks [42], to mention a few.

4. Algorithm for finding the stochastic shortest path

In this section, we propose a DLA-based algorithm for finding a path with the minimum expected length (the shortest path) in a stochastic graph. This algorithm is a modified version of the algorithm given in [6] in which the updating rule for the learning rate has been changed. In this algorithm, the stochastic graph is the random environment for the DLA. We build a DLA based on the given graph. Each node of the DLA corresponds to a node in the graph. The number of actions for every LA in the DLA equals to the out-degree of the corresponding node in the graph. Since the action of a LA represents one of the edges starting from the corresponding node in the graph, hence an action of DLA, which is a sequence of actions of the constituting LAs, represents a particular path in the graph. This algorithm has the following main steps.

1. Choosing a path from the source node to the destination node. This path is chosen by activating the LA corresponding to the source node and choosing one of its actions and activating another LA. This process is continued until a path is traversed. When an edge is traversed, a random length for that edge is assigned. The random length for the edge is taken from the underlying distribution, which is unknown to the algorithm.
2. Computing the length of the chosen path by summing up the edge lengths sampled from the corresponding probability distributions, which are unknown for the algorithm. After choosing a path, its length, and the average length of the traversed paths are calculated.
3. Computing the reinforcement signal and updating the action probability vectors of LAs. The average length of traversed edges is compared with a quantity called the dynamic threshold. Based on the result of this comparison, the reinforcement signal is generated. Then, based on the generated reinforcement signal, the action probability vectors of LAs residing on the chosen path will be updated accordingly.

The above three steps are repeated until the probability of choosing a path becomes greater than a threshold P_{pop} or at least K times the above steps are executed, where P_{pop} and K are two pre-specified values.

The pseudo-code of the modified algorithm is given in Algorithm 1. This algorithm can be described as follows: at first, a DLA isomorphic to the input graph is created (Line 1). After that, the iteration number k is set to 2, and the dynamic threshold is set to a large number (here ∞ to denote a large number) (Line 2). The value of the dynamic threshold is set to a large number because we need to find the minimum of the dynamic threshold and the average value of lengths of traversed paths. To decrease the probability of stocking at local optima, $\bar{L}_1$ is set to the average lengths of some randomly traversed paths (Line 3). This avoids the probability of assigning a small value to $\bar{L}_1$. Then, at iteration k, all LAs in the graph are inactive except the source LA A_s, which chooses an action that activates another LA. This procedure repeated until A_d is reached or no LA can be selected, or the number of activated LAs becomes greater than the number of nodes in the graph (Lines 5-13). When A_d reached, the path denoted by π is created and the length of this path, denoted by L_π, and the average length of sampled paths from v_s to v_d up to now, denoted by $\bar{L}_k$, are computed (Lines 14-20). Then the average length of sampled paths from v_s to v_d is compared with a quantity called the *dynamic threshold*, T_k (Line 16). If $\bar{L}_k$ is less than T_k then all LAs along that path receive a reward and the actions of all LAs are enabled; otherwise, they receive a penalty (Lines 16-19). Based on the received signal, all the LAs along the traversed path update their action probability vectors (Line 18). The destination LA A_d does not need to update its action probability vector because it does not choose any actions. Then, the learning rate is updated using equation (2) (Line 22). The above steps are repeated until the probability of choosing the last path becomes greater than the threshold P_{pop} or K paths are chosen

The L_{R-I} algorithm is employed to update the action probability vectors in the direction from A_d to A_s. In updating phase, the learning rates of LAs at iteration $k \geq 1$ updated using equation (2).

$$a_k = \frac{a}{b + \eta k} \quad \text{for} \quad b \geq 1 > a > 0 \quad \text{and} \quad 0 \leq \eta \leq 1 \tag{2}$$

By using this updating mechanism, the action probability vectors are updated with larger steps in the early stages of the algorithm and this step size decreases as the number of stages increases. This way of updating results in a less switching between different paths, because choosing a path with a very large length doesn't change the shortest path. Dynamic threshold at iteration k, denoted by T_k, equals to the minimum value of sequence $\left(\bar{L}_1, \ldots, \bar{L}_1, \ldots, \bar{L}_{k-1}\right)$. At the beginning of algorithm, T_k is initialized to a large value and $\bar{L}_1$ is set to the average lengths of some randomly traversed paths from v_s to v_d.

Algorithm 1 The pseudo-code of the DLA-based algorithm for solving the stochastic shortest path problem.

Require:

$G = (V, E, \mathcal{F})$ is the input stochastic graph with the source node v_s and the destination node v_d.
P_{pop} is the threshold for path probability,
K is the maximum number of iterations for the algorithm.

Ensure:

The shortest path and the probability that this path is the shortest (*pop*).

1: Construct a DLA isomorphic to the input graph G
2: Set $k \leftarrow 2$ and $T_2 \leftarrow \infty$ ▷ k is the iteration number and T_k is the dynamic threshold at iteration k.
3: Set $\bar{L}_1 \leftarrow$ the average lengths of some randomly traversed paths ▷ $\bar{L}_k$ is the average lengths of traversed paths.
4: **repeat**
5: Set $u \leftarrow v_s$ ▷ Traverse a path starting from the source node.
6: Set $\pi_k \leftarrow \emptyset$ and $L_{\pi_k} \leftarrow 0$ ▷ Initially the set of edges of the kth path is set to empty set and its length to zero.
7: **while** ($u \neq v_d$ **and** $|\pi_k| \leq |V|$) **do** ▷ $|\pi_k|$ is the number of edges in path π_k.
8: A_u selects an action, say α_w and activate the LA, say A_w, on the end of edge (u, w).
9: Add edge (u, w) to the end of path π_k.
10: Set $L_{\pi_k} \leftarrow L_{\pi_k} + C_{uw}$, ▷ Add length of edge (u, w) to the length of the traversed path.
11: Disable action α_w of all unactivated LAs.
12: Set $u \leftarrow w$.
13: **end while**
14: **if** ($u = v_d$) **then**
15: Set $\bar{L}_k = \bar{L}_{k-1} + \frac{1}{k}\left[L_{\pi_k} - \bar{L}_{k-1}\right]$ ▷ $\bar{L}_k$ is the average length of traversed paths from v_s to v_d up to now.
16: **if** ($\bar{L}_k < T_k$) **then**
17: Set $T_{k+1} \leftarrow \bar{L}_k$
18: Update action probability vectors of activated LAs. ▷ Reward the selected actions of activated LAs.
19: **end if**
20: **end if**
21: Set $k \leftarrow k + 1$
22: Update learning rate according equation (2).
23: Enable all actions of all LAs.
24: **until** $\mathbb{P}[\pi_k] \geq P_{pop}$ **or** $k \geq K$
25: **return** The shortest path and the probability that it being the shortest (*pop*).

The procedure of choosing a path from A_s to A_d repeated until the *path probability* denoted by *pop*, is greater than a certain threshold denoted by P_{pop}. The path probability defined as the product of the probability of choosing the actions (edges) of the traversed path. When the algorithm finishes, we hope that the path with the maximum path probability has the minimum expected length among all the paths from source to destination.

In the given input graph, we may have several simple paths and several paths with loops from the source node to the destination node. Since the lengths of all edges are positive, then the algorithm must avoid all paths with loops, because none of these paths are the shortest. To have a simple path, the loops must be excluded. We omit loops by visiting every node along a path being traversed at most once. To do this, when an LA chooses an action, then all unactivated LAs will disable that action from their action sets (Line 11). Disabling an action from the set of actions means that temporarily omitting an action and normalizing the action probability vector, and then after a time, the given action comes back to the action set and the action

probability vector normalized again. This disabling of actions is done by storing the nodes that are traversed in the current path up to now. When an LA is activated, then its actions corresponding to the traversed nodes are disabled by temporarily omitting these actions and normalizing the action probability vectors. During the updating phase, when traveling from the destination node v_d to the source node v_s, the action probability vectors will be updated and then all the disabled actions will be enabled (Line 23). In the rest of this section, we first give a running example and then study the behavior of the proposed algorithm.

4.1. An illustrative example

In order to help the reader to understand the proposed algorithm better, we give a running example of the proposed algorithm on a simple input graph. We consider the graph given in Figure 2 and describe some iterations of the proposed algorithm using this graph. Let $v_s = v_1$ and $v_d = v_4$. Hence, we have five paths from the source node v_1 to the destination node v_4. These five paths are $\pi_1 = (v_1, v_2, v_4), \pi_2 = (v_1, v_2, v_3, v_4), \pi_3 = (v_1, v_4), \pi_4 = (v_1, v_3, v_4), \pi_5 = (v_1, v_3, v_2, v_4)$.

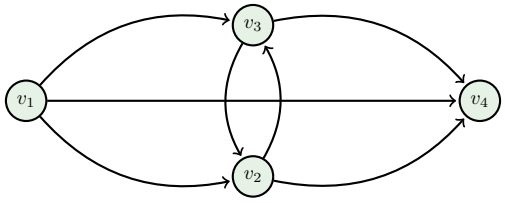

Figure 2: A simple graph as input of the proposed algorithm.

We assume that the length of each edge is a uniformly distributed random variable, where its probability density function is defined as

$$p(x) = \begin{cases} \dfrac{1}{b-a} & \text{if } x \in [a, b] \\ \\ 0 & \text{if } x \notin [a, b] \end{cases} \tag{3}$$

The parameters of distributions for edges' lengths are given in Table 1.

As the Table 1 shows, path $\pi_1 = (v_1, v_2, v_4)$ is shortest path and its expected length equals to 2.

At the first step, a DLA with four nodes and isomorphic to the graph given in the Figure 2 is constructed. Hence, an LA with three actions is assigned to the source node v_1 and two LAs with two actions are assigned to nodes v_2 and v_3, respectively. There is no need to assign any LA to the destination node, because it doesn't select any action, but in DLA, we have the destination LA A_d. At the first step, actions are chosen uniformly. Hence, the action probability vectors are equal to: $\underline{p}^1 = (0.333, 0.333, 0.333)$, $\underline{p}^2 = (0.5, 0.5)$, and $\underline{p}^3 = (0.5, 0.5)$.

Table 1: The distribution of edges' length of graph given in Figure 2.

Edge no	Edge	a	b	μ
1	(1,2)	0	2	1
2	(1,3)	0	2	1
3	(1,4)	0	12	6
4	(2,3)	0	2	1
5	(2,4)	0	2	1
6	(3,2)	0	2	1
7	(3,4)	0	4	2

Let the following setting is used for the algorithm: $\bar{L}_1 = 10$, $a = 0.3$, $b = \eta = 1$. Hence, the learning rate for iteration $k = 2$ equals to $a_2 = 0.1$.

At iteration $k = 2$, the source LA A_1 is activated and assume that it chooses the action α_1^1, and hence the LA A_2 is activated. At this time, a sample length is taken from the distribution of edge $(1, 2)$. Let this sample length be 1.5. Then, assume that the LA A_2 chooses action 2 corresponding to the edge $(2, 3)$, hence LA A_3 will be activated and let edge length 1 will be sampled. LA A_3 has two actions, but one of them creates loop to LA A_2, hence it will be disactivated because the nodes v_1 and v_2 are traversed in the current path up to now. In this case, the probability of action corresponding to the edge $(3, 4)$ will be normalized and its value becomes to 1. Hence, this action will be chosen and let edge length 1.5 will be sampled. Since the destination node is reached the process of selecting actions will be terminated. At this time the length of the selected path is computed, which equals to $L_\pi = 4$ and the average length of traversed paths equals to $\bar{L}_2 = 7$. Since $\bar{L}_2 < T_2$, then we set $T_3 = 7$ and update the action probability vectors of A_1, A_2, and A_3, which become as $\underline{p}^1 = (0.4, 0.3, 0.3)$, $\underline{p}^2 = (0.45, 0.55)$, and $\underline{p}^3 = (0.5, 0.5)$. Then the learning rate will be updated to $a_3 = \frac{0.3}{4} = 0.075$ and a new iteration will be started.

At iteration $k = 3$, the source LA A_1 is activated and let it chooses action α_2^1, and we reach the destination node and let a sample length is taken from the distribution of edge $(1, 4)$. Let this sample length be 8. At this time the length of the selected path is computed, which equals to $L_\pi = 10$ and the average length of traversed paths equals to $\bar{L}_3 = 8$. Since $\bar{L}_3 > T_3$, then we set $T_4 = T_3 = 7$ and no action probability vector is updated. Then the learning rate will be updated to $a_4 = \frac{0.3}{5} = 0.06$ and new iteration will be started.

At iteration $k = 4$, the source LA A_1 is activated and assume that it chooses action α_1^1, and hence LA A_2 is activated. At this time, a sample length is taken from the distribution of edge $(1, 2)$. Let this sample length be 1. Then, let LA A_2 chooses action α_1^2, where the destination node is reached and a sample is taken from the distribution of edge $(2, 4)$. Let this sample length be 1. Since the destination node is reached the process of selecting actions will be terminated. At this time the length of the selected path is computed, which equals to

$L_\pi = 2$ and the average length of traversed paths equals to $\bar{L}_4 = 6.5$. Since $\bar{L}_4 < T_4$, then we set $T_5 = 6.5$ and update the action probability vectors of A_1, and A_2 using learning rate $a_4 = \dfrac{0.3}{1+4} = 0.06$, which become as $\underline{p}^1 = (0.436, 0.282, 0.282)$, and $\underline{p}^2 = (0.483, 0.517)$. Other action probability vectors remain unchanged. Then the learning rate will be updated to $a_5 = \frac{0.3}{6} = 0.05$ and new iteration will be started. This process is repeated until the termination condition is reached.

4.2. Analytical study of the algorithm

In order to study the behavior of the modified algorithm, we study the probability of choosing the shortest path and show that the shortest path will be chosen with a probability as close to unity as desired. In the following theorem, we give the main contribution of the paper, which is the convergence proof of the algorithm.

Theorem 1. *For a graph with unique (single) shortest path from v_s to v_d, Algorithm 1 finds the shortest path with a probability as close to unity as desired.*

Before we prove the main theorem, we first prove the following Lemmas.

Lemma 1. *For sequence $\{a_n\}$ given by equation (2), the following inequalities hold.*

$$\left[\frac{\eta(m-1)+b-a}{\eta n+a}\right]^{\frac{b}{\eta}} \leq \prod_{k=m}^{n}(1-a_k) \leq \left[\frac{\eta m+b}{\eta(n+1)+b-a}\right]^{\frac{b}{\eta}}.$$

Proof. We consider product $\prod_{k=m}^{n}(1-a_k)$. For the first inequality, we have

$$\begin{aligned}\prod_{k=m}^{n}(1-a_k) &= \exp\left[\ln\prod_{k=m}^{n}(1-a_k)\right] \\ &= \exp\left[\sum_{k=m}^{n}\ln(1-a_k)\right] \\ &\geq \exp\left[\int_{m-1}^{n+1}\ln(1-a_k)\right], \end{aligned} \tag{4}$$

and for the second inequality, we have

$$\begin{aligned}\prod_{k=m}^{n}(1-a_k) &= \exp\left[\ln\prod_{k=m}^{n}(1-a_k)\right] \\ &= \exp\left[\sum_{k=m}^{n}\ln(1-a_k)\right] \\ &\leq \exp\left[\int_{m}^{n}\ln(1-a_k)\right]. \end{aligned} \tag{5}$$

Using the convexity property of function $x \ln x$, it follows that

$$(x + \Delta x) \ln(x + \Delta x) - x \ln x \geq \Delta x(1 + x \ln x). \tag{6}$$

Taking into account this inequality and by some algebraic simplification, we obtain

$$\exp\left[\int_{\eta(m)}^{\eta(n)} \ln\left(1 - \frac{a}{x+b}\right)\right] \leq \left[\frac{\eta m + b}{\eta(n+1) + b - a}\right]^{\frac{b}{\eta}}, \tag{7}$$

and

$$\exp\left[\int_{\eta(m-1)}^{\eta(n+1)} \ln\left(1 - \frac{a}{x+b}\right)\right] \geq \left[\frac{\eta(m-1) + b - a}{\eta n + a}\right]^{\frac{b}{\eta}} \tag{8}$$

and hence the Lemma. □

The following corollary considers the case when the learning rate doesn't change.

Corollary 1. *For sequence* $\{a_n\}$ *given by equation* (2)*, when* $\eta = 0$*, then we have*

$$\prod_{k=m}^{n} (1 - a_k) = \left(1 - \frac{a}{b}\right)^{n-m+1} \tag{9}$$

Proof. The proof is trivial by using the Lemma 1. □

Lemma 2. *If repeat–until loop of Algorithm 1 executed infinitely often, then every edge of the graph is sampled infinitely often with a probability as close to unity as desired. In other words, this Lemma asserts that Algorithm 1 samples every edge infinitely often with probability as close to unity as desired.*

Proof. Before we begin the proof of the Lemma, we introduce some definitions and notations. To count the number of times an LA is activated, we define the concept of *local time* for every LA (node). The local time for every LA starts with 1 and incremented by 1 when that LA is activated. Let n_A be the local time for LA A. We now define a sequence of random variables τ_{n_A} ($\tau_1 < \tau_2 < \cdots$), where τ_{n_A} is the global time for LA A activated at n_Ath local time. Let $X_i(m, n)$ to denote the number of times that action α_i of automaton A is selected in the interval $[\tau_{m_A}, \tau_{n_A}]$, $\alpha_i(n)$ to denote the action α_i is chosen by the automaton A at the global time T_{n_A}, $\alpha_{-i}(n)$ to denote that the automaton A chooses action α_j ($j \neq i$) at the global time τ_{n_A}, and $p_i(n)$ to denote the probability of selecting action α_i by A at the global time τ_{n_A}. Then, we show that all successors

of a node, which is sampled infinitely often, is also sampled infinitely often. The proof is by induction on the number of hops(edges) from the source node along a path.

Base step. For the base case, we show that every automaton A_j adjacent to the source automaton A_s is activated infinitely often with probability as close to unity as desired. To do this, we show that

$$\mathbb{P}\left[X_i(m,\infty)=0\right]=0, \tag{10}$$

for all $\tau_{m_A} \leq n \leq \tau_{n_A}$. This means that action α_i is not chosen in time interval $[T_{m_A},\infty]$. If we use the L_{R-I} algorithm, we have

$$\begin{aligned} p_i(n) &= \mathbb{P}\left[\alpha_i(n) \mid a_{-i}(n-1),\ldots,\alpha_{-i}(m),p_i(m)\right] \\ &\geq p_i(m)\prod_{k=m}^{n-1}(1-a_k)\,. \end{aligned} \tag{11}$$

Using inequality (11), we obtain

$$p_i(m) \geq p_i(1)\prod_{k=1}^{m-1}(1-a_k) = D_m, \tag{12}$$

and

$$\begin{aligned} &\mathbb{P}\left[\alpha_{-i}(n) \mid \alpha_{-i}(n-1),\ldots,\alpha_{-i}(m),p_i(m)\right] \\ &= 1-\mathbb{P}\left[\alpha_i(n) \mid \alpha_{-i}(n-1),\ldots,\alpha_{-i}(m),p_i(m)\right] \\ &\leq 1-p_i(m)\prod_{k=m}^{n}(1-a_k)\,. \end{aligned} \tag{13}$$

Also using inequality (11) and the inequality $1-x \leq e^{-x}$, which is valid for all $x \in \mathbb{R}$, we obtain

$$\begin{aligned} &\mathbb{P}\left[\alpha_{-i}(n) \mid \alpha_{-i}(n-1),\ldots,\alpha_{-i}(m),p_i(m)\right] = \\ &\leq \exp\left[-p_i(m)\prod_{k=m}^{n}(1-a_k)\right], \\ &\leq \exp\left[-D_m\prod_{k=m}^{n}(1-a_k)\right]. \end{aligned} \tag{14}$$

Using the chain rule, we also have

$$\begin{aligned}&\mathbb{P}\left[\alpha_{-i}(n),\ldots,\alpha_{-i}(m)\mid p_i(m)\right]=\\&\mathbb{P}\left[\alpha_{-i}(n)\mid\alpha_{-i}(n-1),\ldots,\alpha_{-i}(m),p_i(m)\right]\times\ldots\times\\&\mathbb{P}\left[\alpha_{-i}(m)\mid p_i(m)\right],\end{aligned}\tag{15}$$

and using Lemma 1 and inequalities (12), (14), and (15) we have

$$\begin{aligned}&\mathbb{P}\left[\alpha_{-i}(n),\ldots,\alpha_{-i}(m)\mid p_i(m)\right]\\&\leq\exp\left[-D_m\prod_{k=m}^{n-1}(1-a_k)\right]\times\exp\left[-D_m\prod_{k=m}^{n-2}(1-a_k)\right]\\&\times\ldots\times\exp\left[-D_m\left(1-a_m\right)\right]\\&=\left[1-D_m\right]\exp\left[-D_m\sum_{j=m}^{n-1}\prod_{k=m}^{j}(1-a_k)\right].\end{aligned}\tag{16}$$

Now, using inequalities (11) through (16), we conclude the following inequality.

$$\begin{aligned}\mathbb{P}\left[X_i(m,n)=0\right]&=\mathbb{P}\left[\alpha_{-i}(n),\ldots,\alpha_{-i}(m)\right]\\&\leq\left[1-D_m\right]\exp\left[-D_m\sum_{j=m}^{n-1}\prod_{k=m}^{j}(1-a_k)\right].\end{aligned}\tag{17}$$

So, using Lemma 1 and inequality (17), we obtain that

$$\lim_{n\to\infty}\mathbb{P}\left[X_i(m,n)=0\right]=0,\tag{18}$$

or $\mathbb{P}\left[X_i(m,\infty)=0\right]=0$. Accordingly, we obtain

$$\mathbb{P}\left[X_i(m,\infty)<\infty\right]\leq\sum_{m=1}^{\infty}\mathbb{P}\left[X_i(m,\infty)=0\right]=0,\tag{19}$$

which implies $\mathbb{P}\left[X_i(1,\infty)=\infty\right]=1$.

Inductive step. For the inductive step, we assume that the kth LA along the given path, denoted by A_i is activated infinitely often with a probability as close to unity as desired. Then, we must show that all successors A_j of A_i are also activated infinitely often with a probability as close to unity as desired. Using the fact that

every automaton A_j adjacent to the automaton A_i is activated infinitely often with probability as close to unity as desired (as proved in base step), the proof of induction step is immediate. Since the actions of LAs in nodes of DLA are chosen infinitely often, we conclude that every edge of the graph is sampled infinitely often and hence the proof of Lemma. □

In the following two Lemmas, we show that when every path from the source to the destination nodes is sampled at least M times, then penalty (reward) probabilities are at most at the distance of $\frac{\Delta}{2}$ with high probability. From these Lemmas, we can conclude that for small enough values of Δ or large enough values of M, the environment can be considered to be stationary.

Lemma 3. *Let $q_i(k)$ be the probability of choosing path π_i at iteration k and $\underline{q}(k) = (q_1(k), q_2(k), \ldots)^T$ be the probability of choosing different paths. Define random variable $Y_i(k)$ as the number of times path π_i was chosen up to iteration k. When $\underline{q}(k)$ updated using Algorithm 1, then for every $\delta = \delta(M, K) > 0$, there exists a δ, $a^* \in (0, 1)$, $M > 0$, and $k_0 < \infty$ such that for all $a \in (0, a^*)$ and for all $k > k_0$, the following inequality holds*

$$\mathbb{P}\left[Y_i(k) \geq M\right] > 1 - \delta \qquad \forall\ i = 1, 2, \ldots. \tag{20}$$

Proof. We must show that $\mathbb{P}\left[Y_i(k) \geq M\right] > 1 - \delta$ and equivalently $\mathbb{P}\left[Y_i(k) < M\right] < \delta$. Let us to define another random variable $Z(\pi, k)$ to show that path π is chosen in time k or not. When path π is chosen at time k, we set $Z(\pi, k) = 1$ and otherwise, we set $Z(\pi, k) = 0$. Thus, we have $\mathbb{E}[Z(\pi_i, k)] = q_i(k)$ and variance of $Z(\pi_i, k)$ equals to

$$\begin{aligned} \mathrm{Var}\left[Z(\pi_i, k)\right] &= \mathbb{E}[Z(\pi_i, k)^2] - \mathbb{E}[Z(\pi_i, k)]^2 \\ &= q_i(k)[1 - q_i(k)]. \end{aligned} \tag{21}$$

Since for all pairs of nodes $u \neq w$, events $\{Y_i(k) = u\}$ and $\{Y_i(k) = w\}$ are mutually exclusive, then $Y_i(k) = \sum_{u=1}^{k} Z(\pi_i, u)$. From equations (1) and (2), 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

$$\begin{aligned} \mathbb{E}[Y_i(k)] &= \sum_{s=1}^{k} \mathbb{E}[Z(\pi_i, k)] \geq \sum_{s=1}^{k} q_i(0)(1-a)^s, \\ &= q_i(0)\frac{(1-a) - (1-a)^{k+1}}{a}. \end{aligned} \tag{22}$$

Now, we have

$$\lim_{a\to 0} \frac{(1-a)-(1-a)^{k+1}}{a} = k,$$
$$\lim_{a\to 1} \frac{(1-a)-(1-a)^{k+1}}{a} = 0. \tag{23}$$

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

$$\mathbb{E}\left[Y_i(k)\right] \geq q_i(0)(k-1). \tag{24}$$

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

$$\begin{aligned}
\mathrm{Var}\left[Y_i(k)\right] &= \mathrm{Var}\left[\sum_{u=1}^{k} Z(\pi_i, k)\right] \\
&= \sum_{u=1}^{k} q_i(u)[1-q_i(u)], \\
&\leq \sum_{u=1}^{k} q_i(u) = \mathbb{E}[Y_i(k)].
\end{aligned} \tag{25}$$

Using the Chebyshev inequality, we obtain

$$\begin{aligned}
\mathbb{P}\left[Y_i(k) > M\right] &= 1 - \mathbb{P}\left[Y_i(k) \leq M\right] \\
&\geq 1 - \mathbb{P}\left[|Y_i(k) - \mathbb{E}[Y_i(k)]| \geq \mathbb{E}[Y_i(k)] - M\right] \\
&\geq 1 - \frac{\mathrm{Var}\left[Y_i(k)\right]}{\left(\mathbb{E}[Y_i(k)] - M\right)^2} \\
&= 1 - \frac{\mathbb{E}[Y_i(k)]}{\left(\mathbb{E}[Y_i(k)] - M\right)^2}.
\end{aligned} \tag{26}$$

Inequality (24) shows that $\mathbb{E}[Y_i(k)]$ increases linearly with k. Hence, 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

$$\mathbb{P}\left[Y_i(k) \geq M\right] \geq 1 - \delta. \tag{27}$$

Now considering all paths Π, there exists an $a^* \in (0,1)$ and a $k_o < \infty$ such that

$$k_o = \max_{\pi}\{k_o(\pi, k)\}$$
$$a^* = \min_{\pi}\{a(\pi, k)\}. \tag{28}$$

Hence,

$$\mathbb{P}\left[Y_i(k) > M\right] \geq 1 - \delta, \tag{29}$$

for all $a \in (0, a^*)$ and for all $k > k_o$. This completes the proof of Lemma. □

Lemma 4. *For a stochastic graph with unique (single) shortest path, let path π_i is chosen in iteration k. Also let $c_i(k) = \mathbb{P}\left[\bar{C}_k > T_k\right]$, $c_i^* = \lim_{k\to\infty} c_i(k)$, and let $Y_i(k)$ be the number of times path π_i is selected up to iteration k. Then for any $\epsilon > 0$ and $\delta \in (0,1)$, we have*

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

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

Proof. Let Δ be the difference between two smallest elements of $\{c_1^*, \ldots, c_r^*\}$. Since the graph has only one shortest path from the source node to the destination node, then $\Delta > 0$. Let X_k be the indicator function such that $X_i(k) = 1$ when at iteration k, path π_i was chosen and actions constituting path π_i are rewarded. Weak law of large numbers states that for a given $\delta > 0$, there exists $k_i(\Delta, \delta) < \infty$ such that when path π_i is chosen at least $k_i(\Delta, \delta)$ times ($k_i(\Delta, \delta) < Y_i(k)$, we have

$$\mathbb{P}\left[|c_i^* - c_i(k)| < \frac{\Delta}{2}\right] > 1 - \delta. \tag{30}$$

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

$$\mathbb{P}\left[c_j(k) > c_i(k)\right] > 1 - \delta. \tag{31}$$

By Lemma 3, we know that we find values for k_o and a^* such that

$$\mathbb{P}\left[\min_i\{Y_i(k)\} > M\right] > 1 - \delta, \tag{32}$$

for all $k > k_o$ and all $a \in (0, a^*)$. Thus if all paths are chosen at least M times, each $c_i(k)$ will be in a $\Delta/2$ neighborhood of c_i^* with an arbitrary large probability and hence Lemma. □

Lemma 5. *If the dynamic threshold T_k is updated using Algorithm 1, then the given algorithm finds the shortest path.*

Proof. In Algorithm 1, $\bar{L}_k$ represents the average length of paths which are sampled up to iteration k. Suppose that l_{min} (l_{max}) denotes the average length of paths with minimum (maximum) expected value, thus we have $l_{min} \leq \bar{L}_k \leq l_{max}$. From lines 16 through 19 of Algorithm 1, we have $T_{k+1} \leq T_k$. From Lemma 2, every source–destination path is sampled infinitely often including the paths with lengths less than $\bar{L}_k$. Therefore, line 17 is executed infinitely often until the the shortest path found, that is $T_k = l_{min}$. Define local time m for the segment containing lines 17 through 18, the value of m is incremented by 1 each time this block is executed. So we have $T_{m+1} < T_m$. Since T_m is a monotonically decreasing function with minimum value of l_{min} we have

$$\lim_{m \to \infty} T_m = l_{min},$$

and hence the Lemma. □

Proof of Theorem 1. The $\bar{L}_k$ at iteration k is

$$\bar{L}_k = \sum_{i=1}^{t} q_i(k) \bar{L}_i(k)$$

where $q_j(k)$ is the probability of choosing path π_j at iteration k and $\bar{L}_j(n)$ is the average length of path π_j up to iteration k. Using Lemma 5, when every path from the source node to the destination node is sampled at least M times, then the penalty (reward) probabilities are at most at the distance of $\frac{\Delta}{2}$ with high probability. Hence, for small enough values of Δ or large enough values of M, we can consider that the random environment is stationary and from our assumption that the graph has only one shortest path, hence the DLA has a single optimal action. Then, using the convergence results of the L_{R-I} algorithm, it can be concluded that the DLA selects its optimal action with probability as close to unity as desired. Therefore, according to Lemma 4, we have $\lim_{m\to\infty} T(m) = \lim_{m\to\infty} \bar{L}_m = l_{min}$, which is obtained when the DLA selects its optimal action (the shortest path). This condition is satisfied when $\lim_{m\to\infty} q_i(m) = 1$. So for a graph with only one shortest path, the probability of choosing the shortest path converges to unity as desired. □

These results show that by using Algorithm 1 for updating the action probability vectors, this algorithm

finds the shortest path with a high probability, which can be close to unity as possible as.

5. Experiments

In this section, we evaluate the performance of the given algorithm using different stochastic graphs. In this section, we first give the graphs used for evaluating the algorithm and then give the performance measures being used for evaluation and finally give the experimental results.

5.1. Graphs used for evaluation

For evaluating the proposed algorithm, we used three types of graphs: graphs with predefined structure, random graphs, and Kronecker graphs. In the experiments, we used the following three stochastic graphs with a predefined structure. These graphs are borrowed from [15].

1. The first graph is shown in Figure 3. This graph has ten nodes and twenty-three edges. The source node has label 1 and the destination node has label 10. The label of each edge is the mean value of length distribution for the corresponding edge, which has an exponential distribution. For instance, the length of edge $(1, 2)$ has the exponential distribution with a mean of 12.

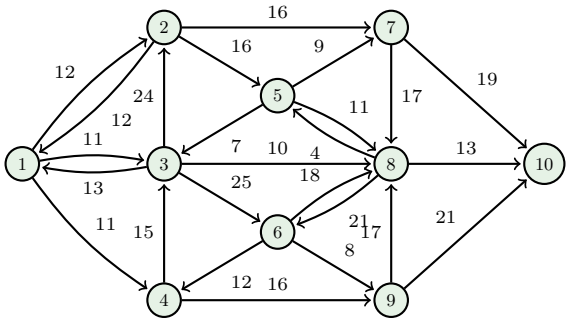

Figure 3: Graphs 1 and 2

2. The second graph, which is shown in the Figure 3, has the same structure as the first graph but with different edge length distribution. The distribution of edge lengths for this graph is given in Table A.8. In this graph, the source node is the node labeled $'1'$ and the destination node labeled by $'10'$.
3. The third graph is shown in the Figure 4 and the length distribution of edges is given in Table A.9. This graph has fifteen nodes and forty-two edges. In this graph, the source node has label $'1'$, and the destination node has label by $'15'$.

The second type of graphs is a group of random graphs where the number of nodes is in the range of 10 nodes to 1000 nodes. These graphs are generated in the following way. For a given pair of nodes, first, with

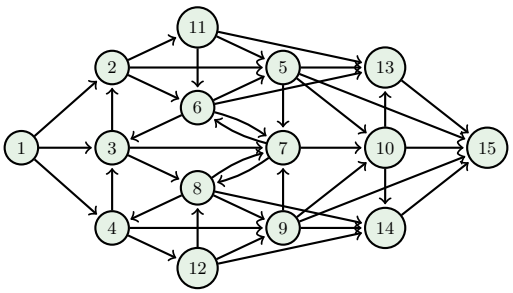

Figure 4: Graph 3

a small probability, an edge is created and then the length of this edge is specified. In these graphs, the edge lengths are normally distributed random variables. The mean value of each edge length is uniformly distributed random variable in the interval $[6, 200]$. The variance of the normal density functions is set to 3. Finally, two nodes are chosen as the source node and the destination node, respectively.

The third type of graphs is a group of Kronecker graphs. Stochastic Kronecker graph model is an easy way to generate synthetic graphs, which are similar to natural graphs [43]. The core of this model is a simple matrix operation, which is called the Kronecker product. The Kronecker product is an operation on two matrices. This operation nests many copies of the second matrix within the first matrix. This generation model produces a self-similar graph. The stochastic Kronecker graphs are generated by taking successive Kronecker powers of an initial stochastic adjacency matrix. After generating a stochastic Kronecker graph, we assign a length to each edge. The distribution of lengths is generated in the same way as the random graphs.

5.2. Performance measures

To evaluate different algorithms, we used the following four performance measures. The first performance measure is the average number of iterations (AVI). This performance measure shows the average number of iterations needed for the algorithm to find the shortest path. When the algorithm fails to find the shortest path, the number of iterations used by the algorithm to terminate is used to calculate the average number of iterations. A smaller value for AVI is better.

The second performance measure is the percentage of converged runs (PC). Since both the proposed algorithm and the input graphs are random, we need to run the algorithm several times and then study the behavior of the algorithm on these runs. This performance measure shows what percentage of runs finds the shortest path correctly. A larger value for PC is better.

The third performance measure is the average number of samples (TS) taken from the edges of the graph. This performance measure shows that on average how many samples are taken from the edges of the graph to find the shortest path. A smaller value for TC is better.

The fourth performance measure is the average number of samples taken from the edges of the shortest path (SPS). This performance measure shows that on average how many samples are taken from the edges along the shortest path. Thus, we have $SPS \leq TS$. A smaller value of the difference between TS and SPSS shows that the algorithm intelligently takes more samples for the edges along the stochastic shortest path.

5.3. Results for graphs with predefined structure

The algorithm given in this paper and algorithms given in [5, 6, 7] are executed 1100 times on every graph of the graphs 1 through 3 and the results of experiments for each graph are summarized in Table 2. The two first columns for every algorithm contains the average number of iterations (AVI) for runs which are converged and the percentage of converged runs (PC). The next two columns for every algorithm contains the average number of samples (TS) taken from the edges of the graph and the average number of samples taken from the edges of the shortest path (SPS). For all algorithms the following settings are used: P_{pop} is set to 0.9, the maximum number of traversed path (K) is set to $900,000$, and the learning rate is changed in the interval $[0.006, 0.007]$ with step length of 0.0001, and for the proposed algorithm we set b is set to 1, η is set to 1.

Table 2: The experimental results of different algorithms on graphs with predefined structures.

	This algorithm				Algorithm [6]				Algorithm [7]				Algorithm [5]			
Graph	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS
Graph 1	1895	99.63	4315	1633	**1751**	78	**4213**	**1613**	3204	99	11672	7353	3683	100	13552	5862
Graph 2	2356	99.18	8724	4104	**2062**	94	**6776**	**2802**	6217	100	21251	11461	8547	99.81	26081	12182
Graph 3	28067	98.63	36380	18822	**3275**	70	**15192**	**4814**	7805	98	38541	23427	13133	100	42004	24680

The results reported in Table 2, show that the proposed algorithm requires almost the same number of samples for the graphs 1 and 2 than the algorithms given in [6] but at the higher rate of convergence. For graph 3, the rate of convergence of the proposed algorithm is increased at the cost of increasing the number of samples. For graphs 1 and 2, the proposed algorithm requires a smaller number of samples than the algorithms given in [5, 7] at almost the same rate of convergence and for graph 3, it requires more samples than the algorithms given in [5, 7]. The experimental results given in these Tables show that the input graph has a large effect on the performance of all the mentioned algorithms.

5.4. Results for random graphs

The algorithm given in this paper and algorithms given in [5, 6] are executed 100 times on every random graph. Table 3 shows the results of the modified algorithm on the random graphs. The following settings are used in this experiments: a is set to 0.01, b is set to 1, η is set to 1, P_{pop} is set to 0.9 and the maximum number of traversed path (K) is set to $900,000$. Table 3 also shows that for most graphs, the algorithm finds the shortest path. The same conclusion as given in subsection 5.3 can be made here for the random graphs.

Table 3: The results of the modified algorithm on the random graphs.

	This algorithm				Algorithm [6]				Algorithm [7]				Algorithm [5]			
n	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS
10	708	100	2087	1521	695	99	1981	1472	3254	100	5175	3863	3312	100	5349	3928
20	717	100	2880	1515	722	98	2930	1412	3314	100	4987	3784	3475	100	5024	3997
50	753	100	2436	1559	965	100	14340	7977	2262	100	4473	2253	1933	100	3643	1928
100	625	100	1786	1311	933	97	5819	3263	2132	100	13594	7567	2489	100	17079	8885
200	889	99	6591	3500	844	96	3287	831	2324	100	6793	2311	1919	100	5384	1897
300	804	96	16021	2847	836	94	3301	829	2227	98	7816	2212	1920	99	6929	1899
400	772	96	12668	2785	956	99	15403	3558	2579	100	32179	9272	2189	100	27961	7882
500	741	100	6052	1579	1004	97	18057	3645	2213	100	33226	8147	2538	100	36209	9291
600	1006	96	10505	1833	1059	90	10423	2234	2230	97	21602	4721	2639	98	28124	5608
700	801	97	18731	3652	984	94	28745	5115	2248	99	48306	11138	2266	98	35052	10003
800	778	100	16223	2876	984	97	20857	3533	2243	100	43491	8195	2606	100	48551	9702
900	262	100	5180	1412	915	96	9918	4152	2078	100	19434	9438	2429	100	23002	11001
1000	935	100	29065	4936	942	92	29172	4981	3598	100	63218	8318	3799	100	65849	8488

5.5. Results for stochastic Kronecker graphs

The algorithm given in this paper and algorithms given in [5, 6] are executed 100 times on every stochastic Kronecker graph. Table 4 shows the results of the modified algorithm on the random graphs. The following settings are used in this experiments: a is set to 0.01, b is set to 1, η is set to 1, P_{pop} is set to 0.9 and the maximum number of traversed path (K) is set to $900,000$. Table 4 also shows that for most graphs, the algorithm finds the shortest path.

Table 4: The results of the modified algorithm on the stochastic Kronecker graphs.

	This algorithm				Algorithm [6]				Algorithm [7]				Algorithm [5]			
n	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS
10	828	100	1783	1423	831	100	1202	740	1195	100	1960	939	1063	100	1605	946
20	769	100	1450	674	923	100	2727	1884	1650	100	4994	3348	1821	100	5695	3770
50	622	100	1049	621	776	100	1255	775	1777	100	2881	1775	1553	100	2492	1552
100	738	100	5348	1648	921	100	6286	2071	2459	100	17626	5571	2121	100	12961	4655
200	779	100	11034	3722	969	100	14142	4614	1948	100	26159	9209	2282	100	28473	10718
300	860	100	13988	4088	1094	99	21334	5529	2044	100	26949	9238	2179	100	33086	10531
400	1159	99	12364	3842	1175	99	13592	3967	2037	100	26098	7308	1981	100	27513	7464
500	798	98	28423	4791	873	95	4948	1718	1043	100	11982	3679	1094	100	12199	3793
600	1188	96	30009	5304	839	94	27731	4473	1409	97	35188	6907	1234	96	23253	5587

5.6. The effect of graph structure on the performance

The previous experiments on graphs 1 through 3, random graphs, and stochastic Kronecker graphs show that when a increases, the average number of iterations needed for the algorithm to find the shortest path decreases. This is because of the amount of updates to the action probabilities become larger and the path probability reaches the P_{pop} faster. It seems that when the graph becomes larger, then the average number of iterations also increases. But Tables 3 and 4 don't confirm this. To study this issue, we set up two other experiments: the first one studies the average out-degree of nodes and the second one studies the number of edges along the shortest path on graphs with 50 nodes. The settings of the previous experiment are used for this experiment except $a = 0.001$. The effect of the average out-degree of the graph on the performance of the algorithm is reported in Table 5. Table 5 also shows that when the average out-degree becomes larger, then the average number of iterations increases, and the percentage of convergence decreases. This is because when

average out-degree increases, the number of source-destination paths is increased, and also the initial values of probabilities of choosing different paths are decreased. Hence, by increasing the average out-degree, the percentage of convergence rate decreases. The reason is that the number of paths from v_s to v_d is increased and as a result, the initial probabilities of choosing different paths are decreased. Hence, the probabilities of choosing paths are updated by a smaller amount, which increases the number of iterations to reach P_{pop}

Table 5: The effect of average out-degree of the graph on the performance of the algorithm.

Average	This algorithm				Algorithm [6]				Algorithm [7]				Algorithm [5]			
out degree	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS
3	2014	100	7540	1917	1531	99	8594	4695	2908	100	22152	6051	2915	100	11781	8270
3.78	7838	100	37410	12336	2406	91	10164	2398	4083	100	14414	7835	3020	100	11571	5371
5.1	9842	99	36584	12250	8568	95	41746	10796	28676	100	167959	36765	24566	100	117196	28248
6.3	20498	100	80399	22283	21814	98	83604	32729	30908	100	98703	46194	31423	100	109730	30426
6.56	26131	98	102776	25465	21163	94	84571	34027	29580	99	138352	55866	49731	100	201759	73353

The effect of the number of edges along the shortest path on the performance of the algorithm is reported in Table 6. Table 6 also shows that when the number of edges becomes larger, then the average number of iterations increases. This is because when the number of edges in the paths from v_s to v_d is increased, the number of source-destination paths is increased, and also the initial values of probabilities of choosing different paths are decreased. By increasing the number of edges, the algorithm needs more time to update path probability to reach P_{pop}, because the probabilities are updated by smaller values.

Table 6: The effect of the number of edges along the shortest path on the performance of the algorithm.

Number	This algorithm				Algorithm [6]				Algorithm [7]				Algorithm [5]			
of edges	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS	AVI	PC	TS	SPS
1	2014	100	7540	1917	1690	100	3607	1634	6625	100	13346	6541	6699	100	12901	6544
2	2367	100	37410	5760	2343	100	7574	5079	6215	100	19888	13265	7184	100	23016	15282
3	3535	100	47300	13494	3151	98	29723	10400	6664	100	53212	24005	7938	100	68184	28239
4	3908	100	65591	22283	3644	95	45272	19441	7065	100	87108	38062	8382	100	69301	37918
5	6223	99	102696	50926	6297	94	73309	44236	10163	98	127974	81093	11001	99	118223	71641

5.7. Any time behavior of the proposed algorithm

A property of this algorithm is *any time behavior*. An algorithm called anytime if it can produce a valid solution to a problem before its termination [44]. We study this property by plotting the probability of selecting the shortest path with running time (see Figure 5). A point (x, y) at this curve is the probability (y) of obtaining a solution if the algorithm is allowed to run for x iterations. This curve can be divided into two approximately equally long parts. In the first part, the curve grows faster than in the second part. This fact suggests that it is not necessary to have very long runs and more efforts may not provide a better solution quality. This property holds for many iterative improvements algorithms. The average POP of taken over one hundred converged runs for different algorithms are given in the Figure 5. Note that a point (x, y) in this curve shows the average number of iterations x needed for the algorithm to find the shortest path with the probability of y. Comparing

the results of different algorithms, Figure 5 shows that for a given convergence rate, the modified algorithm requires fewer samples from the edges of the graph and hence the less computation time.

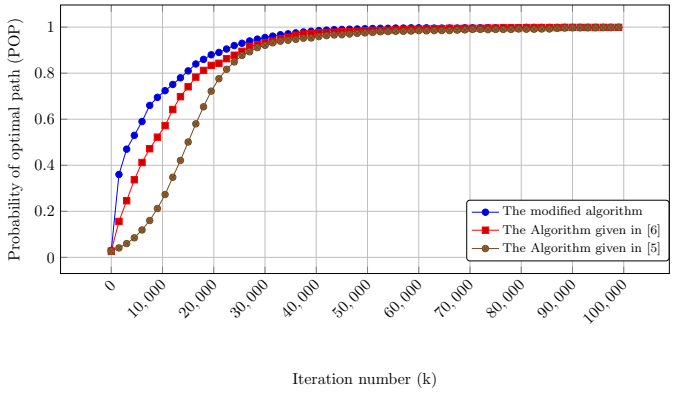

Figure 5: Probability of the shortest path for graph 2

In the rest of this section, we study the anytime behavior of the algorithm with respect to the number of nodes of the graph, the average out-degree of nodes in the graph, and the number of edges along the shortest path.

5.7.1. Effect of the number of nodes

Figure 6 shows the probability of choosing the shortest path for a typical run of the algorithm as the algorithm iterates for different random graphs. Figure 6 also shows that the number of iterations doesn't depend on the number of nodes of the graph and other important factors affect the number of iterations. These results also show that the algorithm can be scaled up.

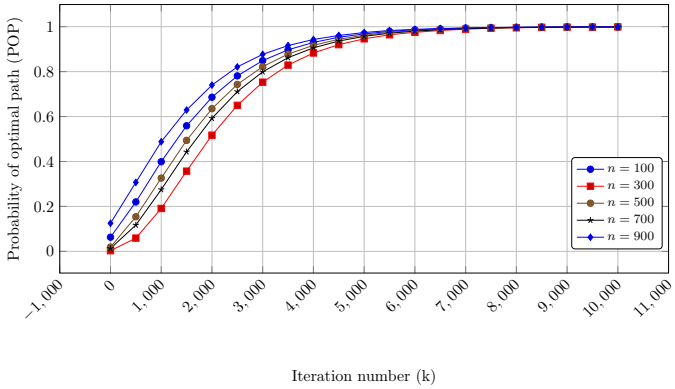

Figure 6: Probability of the shortest path for graph a random graph with different number of nodes (n).

5.7.2. The effect of average out-degree of nodes

Figure 7 shows the probability of choosing the shortest path for a typical run of the algorithm as the algorithm iterates for the different average out-degree of nodes in the random graphs with 50 nodes. Figure 7

also shows that the slope of the curve decreases as the average out-degree of nodes in the graph increases. Hence, the algorithm finds the shortest path more quickly for graphs with small degrees. This is because the initial probabilities of action are larger and updated with larger values. Hence, the number of iterations (samples) will be decreased as the out-degree of the nodes in the graph decreased and the algorithm needs the smaller number of samples to find the shortest path.

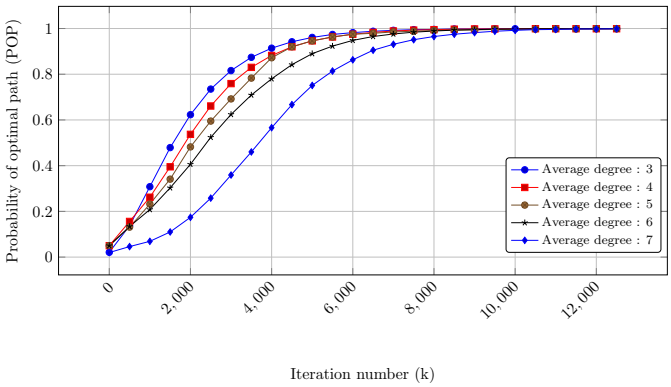

Figure 7: Probability of the shortest path for graph a random graph with 50 nodes and different average out-degrees.

5.7.3. The effect of the number of nodes along the shortest path

Figure 8 shows the probability of choosing the shortest path for a typical run of the algorithm as the algorithm iterates for a different number of nodes along the shortest path in the graph. Figure 8 also shows that the slope of the curve decreases as the number of nodes along the shortest path in the graph increases. The reason is that the path probability (*pop*) inversely depends on the number of edges in the shortest path. As the number of edges along the shortest path increases, the initial value of the path probability decreases and updated with smaller values. Hence, the number of iterations (samples) will be increased as the number of edges along the shortest path increased and the algorithm needs more samples to find the shortest path.

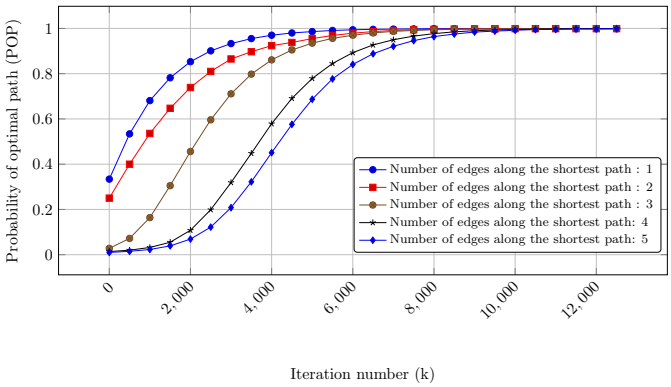

Figure 8: Probability of the shortest path for graph a random graph with 50 nodes and different number of edges along the shortest path.

5.8. Discussions

In the previous subsections, we reported the results of experiments on different graphs. By carefully inspection on the Table 2, it is apparent that two graphs 1 and 2 have the same structure but different distributions for the length of edges. In these two graphs, the expected lengths of different paths are different. Since, the algorithm tries to find a path with the minimum penalty probability, when the difference between penalty probabilities are small, the LA needs to have more interactions with the random environment to find the action with the minimum penalty probability. Specifically, when two smallest penalty probabilities are almost the same, we have the same problem. In the SSPP, this is the same as when the difference between the expected lengths of the shortest paths and the second shortest path becomes small. Hence, one important factor is the difference between the expected lengths of different paths. This is the reason for the difference between the two first rows of the Table 2. By analyzing the Tables 5 and 6, it is apparent that two other important factors that affect the performance of the algorithms are the average degree of nodes in the graph and the number of the edges along the shortest path in the graph. The reason is that the initial action probabilities of LAs become small and these probabilities will be updated with small values. Hence, the average iterations for finding the shortest paths will be increased, which also increases the number of samples taken from the edges. Analyzing the Tables 3 and 4 reveals that increasing the number of nodes in the graph does not necessarily increase the average number of iterations for the algorithm to find the shortest path. This result can also be seen in the Figure 6. As a result, the following factors are very important in the SSPP: the difference between the lengths of the shortest path and the second shortest path, the average out-degrees of nodes, and the number of edges along the shortest path.

6. Computational complexity of the algorithms

In this section, we study the computational complexity of the following four algorithms: the proposed algorithm, the three algorithm given in [5, 6, 7]. The computation times of these four algorithms are the order of $O(rkt)$, where r is the number of actions of LA, k is the time needed to select a path and update the action probability vectors for that path, and t is the number of iterations that algorithm needs to find the shortest path. Hence, the computation time of the algorithms depends on the three factors: The first one is the time to select an action and update the corresponding action probability vector, which is in order of $O(r)$. The number of actions (r) for each LA equals to out-degree of the corresponding node and usually is small. This time for the proposed algorithm and the algorithms given in [5, 6] is the same because they use the same learning algorithm and the same information in each LA. But the algorithm given in [7] uses more information in each node and

needs more computation. Hence, this algorithm needs more time to update action probability vectors, but the order of time complexity for all four-mentioned algorithms are the same. The second one is the number of times that paths are selected and the probability vectors are updated. This time depends on the number of edges in the selected path. The exact value of this time can't be determined but its bound can be determined. The upper bound on the number of edges in a path is equal to n, where n is the number of nodes in the graph. Hence, we have $k = O(n)$. The last one is the number of iteration needed by the algorithm to find the shortest path. This number is different for each algorithm and specifies the computation time of the algorithm. We can also specify the value of kt, which is the number of times that LAs select their actions. This equals to the number of samples taken from the edges of the graph and in section 5, we denoted by TS. Since when each LA selects an action, it means that an edge must be sampled. Then, the number of times that actions of LAs are selected equals the number of samples taken from the edges of the graph. Hence, the algorithm which needs a smaller number of samples has smaller computational costs for these two parts. A factor that also affects the computational time is the number of times that action probability vectors are updated. This time depends on how dynamic threshold changes. This time can't be analyzed analytically, but when an algorithm converges faster, the algorithm updates the action probability vectors on the shortest path more. This time is proportionally dependent on the average iterations needed to find the shortest path. Hence, the algorithm that needs a smaller value of average iterations and the smaller number of samples runs faster. Experimental results show that the modified algorithm also needs a lower computational time than the other three algorithms (for graphs 1 and 2). As a conclusion, all the above mentioned algorithms have the same order of computational complexity.

P2.2 To compare the exact running times of algorithms, we implemented all algorithms in Phyton language and compared their running using 10 different generated random graphs. For each graph, we run all algorithms 100 times and then average the running times of all different algorithms on these 1000 different and independent runs. The running time for different algorithms are given in Table 6, which shows that the running times of all the mentioned algorithms are in the same order. For each edge sample, the algorithm needs to select an action and update an action probability vector. The algorithm proposed in this paper and the algorithms given in [5, 6] need almost the same running time per action selection and updating the action probability vector for a given node. Since the number of times actions selected and the number of time action probability vector updated equal to the total number of samples taken from the edges of the given graphs, hence the algorithm that needs the smaller number of samples taken from the edges also needs the smaller running times. In these random graphs, the algorithm proposed in this paper needs the smaller number of samples taken from the edges

and then the algorithms given in [6] and [5] have the next ranks, respectively. The algorithm given in [7] needs the largest computation for selecting actions and updating action probability vectors, but it needs a smaller number of samples taken from the edges than the algorithm given in [5]. But the overall running time is higher than the other three algorithms due to the higher computation time per sample.

Table 7: Average running time of algorithms to find the shortest path in a random graph with 100 nodes.

Algorithm	Running time (ms)
The proposed algorithm	863.37
Algorithm given in [5]	881.07
Algorithm given in [6]	875.52
Algorithm given in [7]	891.90

7. Conclusions

In this paper, we studied a DLA-based algorithm for finding the shortest path in stochastic graphs. The method given in this paper provides a way that can be used to choose a path from the source node to the destination node with minimal expected length. A new method of proof is proposed for an arbitrary graph, which can be used for an arbitrary input graph. It is shown that when all LAs of DLA use a L_{R-I} algorithm, the modified algorithm finds the shortest path with probability as close to unity as desired. Some experiments are designed to study the effect of the number of nodes, the average out-degree of nodes, and the number of edges along the shortest path on the performance of the algorithm. The experimental results show that by increasing these three parameters, the average number of iterations and the average number of samples taken from the graph, which increases the computation time of the algorithm. In future works, we try to have a finite-time analysis of the DLA and specially finite-time analysis of the given algorithm.

Acknowledgments

The authors would like to thank anonymous reviewers for their time, valuable comments, constructive criticism, and suggestions which greatly improved the paper. The authors also would like to thank Mr. Mahmoud Karimian for his help.

References

[1] L. Fu, L. R. Rilett, Expected Shortest Paths in Dynamic and Stochastic Traffic Networks, Transportation Research: Part B-Methodological 32 (1998) 499–516.

[2] G. H. Polychronopoulos, J. N. Tsitsiklis, Stochastic Shortest Path Problems with Recourse, Networks 27 (1990) 133–443.

[3] H. Frank, Shortest Path in Probabilistic Graphs, Opertions Research 15 (1969) 583–599.

[4] P. B. Mirchandani, H. Soroush, Shortest Distance and Reliability of Probabilistic Networks: A Case with Temporary Preferences, Computer Operations Research 13 (4) (1985) 365–087.

[5] H. Beigy, M. R. Meybodi, Utilizing distributed learning automata to solve stochastic shortest path problems, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 14 (5) (2006) 591–615.

[6] H. Beigy, M. R. Meybodi, An iterative stochastic algorithm based on distributed learning automata for finding the stochastic shortest path in stochastic graphs, Journal of Supercomputing (2020) 1–23.

[7] M. R. M. Meybodi, M. R. Meybodi, Extended distributed learning automata: An automata-based framework for solving stochastic graph optimization problems, Applied Intelligence 41 (3) (2014) 923–940.

[8] S. M. Vahidipour, M. R. Meybodi, M. Esnaashari, Finding the shortest path in stochastic graphs using learning automata and adaptive stochastic petri nets, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 25 (3) (2017) 427–455.

[9] S. Vahidipour, M. Esnaashari, A. Rezvanian, M. Meybodi, GAPN-LA: A framework for solving graph problems using petri nets and learning automata, Engineering Applications of Artificial Intelligence 77 (2019) 255–267.

[10] A. A. B. Pritsker, Application of Multi-Channel Queuing Results to the Analysip of Conveyor Systems, Journal of Industrial Engineeriig (1966) 14–42.

[11] J. J. Martin, Distribution of the Time Through a Directed Acyclic Network, Operations Research (1965) 60–56.

[12] C. U. Sizal, V. A. Q. Pritsker, J. J. Solberg, The Use of Cutsets in Monte-Carlo analysis of Stochastic Networks, Math. Comp. Simulation 21 (1979) 276–384.

[13] P. B. Mirchandani, Shortest Distance and Reliability of Probabilistic Netwnrks, Computer Operations Research 8 (1970) 347–315.

[14] C. C. Sigal, A. A. B. Pritsker, J. J. Solberg, The Stochastic Shortest Route Problem, Operations Research 48 (1980) 1422–1199.

[15] C. Alexopoulos, State Space Partitioning Methods for Stochastic Shortest Path Problems, Networks 30 (1997) 9–21.

[16] E. Miller-Hooks, H. Mahmassani, Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks, Transportation Science 34 (2000) 198–215.

[17] E. Miller-Hooks, Adaptive Least-Expected Time Paths in Stochastic, Time-Varying Transportation and Data Networks, Networks 37 (1) (2001) 35–52.

[18] S. T. Waller, A. K. Ziliaskopoulos, On the Online Stochastic Shortest Paths with Limited Arc Cost Dependencies, Networks 40 (4) (2002) 216–227.

[19] Y. Y. Fan, R. E. Kalaba, J. E. Moore, Shortest Paths Stochastic Networks with Correlated Link Costs, Computer and Mathematics with Applications 49 (2005) 1549–1564.

[20] S. K. Peer, D. K. Sharma, Finding the Shortest Path in Stochastic Networks, Computer and Mathematics with Applications 53 (2007) 729–740.

[21] B. Thomas, C. White, The Dynamic Shortest Path Problem with Unticipation, European Journals of Operation Research 176 (2) (2007) 836–854.

[22] A. Das, C. Martel, Stochastic Shortest Path with Unlimited Hopes, Information Processing Letters 109 (2009) 290 295.

[23] F. Teichteil-Königsbuch, Stochastic safest and shortest path problems, in: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, 2012, pp. 1825–1831.

[24] L. E. Pineda, K. H. Wray, S. Zilberstein, 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., 2017, pp. 3629–3635.

[25] M. Guillot, G. Stauffer, The stochastic shortest path problem: A polyhedral combinatorics perspective, European Journal of Operational Research 285 (1) (2020) 148–158.

[26] W. Zeng, T. Miwa, Y. Wakita, T. Morikawa, Application of lagrangian relaxation approach to α-reliable path finding in stochastic networks with correlated link travel times, Transportation Research Part C: Emerging 56 (3) (2015) 309–334.

[27] B. Y. Chen, C. Shi, J. Zhang, W. H. K. Lam, Q. Li, S. Xiang, Most reliable path-finding algorithm for maximizing on-time arrival probability, Transportmetrica B: Transport Dynamics 5 (3) (2017) 1–17.

[28] S. Misra, B. J. Oommen, Using Pursuit Automata for Estimating Stable Shortest Paths in Stochastic Network Environments, International Journal of Communication Systems 22 (2009) 441–468.

[29] O. V. Nedzelnitsky, K. S. Narendra, Nonstationary Models of Learning Automata Routing in Data Communication Networks, IEEE Transactions on Systems, Man, and Cybernetics SMC-17 (6) (1987) 1004–1015.

[30] B. J. Oommen, E. V. de St. Croix, Graph Partitioning Using Learning Automata, IEEE Transactions on Computers 45 (2) (1996) 195–208.

[31] B. J. Oommen, T. D. Roberts, Continuous Learning Automata Solutions to the Capacity Assignment Problem, IEEE Transactions on Computers 49 (6) (2000) 608–620.

[32] H. Beigy, M. R. Meybodi, Adaptation of Parameters of BP Algorithm using Learning Automata, in: Proceedings of VI Brazilian Symposium on Neural Networks, SBRN2000, Brazil, 2000, pp. 24–31.

[33] H. Beigy, M. R. Meybodi, A learning automata-based algorithm for determination of the number of hidden units for three layer neural networks, International Journal of Systems Science 40 (1) (2009) 101–118.

[34] B. Moradabadi, M. R. Meybodi, Link prediction in stochastic social networks: Learning automata approach, Journal of Computational Science 127 (2017).

[35] B. Moradabadi, H. Beigy, A new real-coded bayesian optimization algorithm based on a team of learning automata for continuous optimization, Genetic Programming and Evolvable Machines 15 (2) (2014) 169–193.

[36] H. Beigy, M. R. Meybodi, Cellular learning automata based dynamic channel assignment algorithms, International Journal of Computational Intelligence and Applications 8 (3) (2009) 287—314.

[37] H. Beigy, M. R. Meybodi, Adaptive limited fractional guard channel algorithms: A learning automata approach, International Journal of Uncertainty, Fuzziness and Knowledge-based Systems 17 (6) (2009) 881—913.

[38] M. Ahangaran, N. Taghizadeh, H. Beigy, Associative cellular learning automata and its applications, Applied Soft Computing 53 (2017) 1–18.

[39] A. M. Saghiri, M. D. Khomami, M. R. Meybodi, Intelligent Random Walk: An Approach Based on Learning Automata, Springer, 2019.

[40] M. Hasanzadeh, M. R. Meybodi, Grid resource discovery based on distributed learning automata, Computing 96 (9) (2014) 909–922.

[41] A. Rezvanian, M. R. Meybodi, Finding maximum clique in stochastic graphs using distributed learning automata, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 23 (1) (2015) 1–32.

[42] B. Moradabadi, M. R. Meybodi, Link prediction in fuzzy social networks using distributed learning automata, Applied Intelligence 47 (3) (2017) 837–849.

[43] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, Z. Ghahramani, Kronecker graphs: An approach to modeling networks,

Journal of Machine Learning Research 11 (33) (2010) 985–1042.

[44] S. Zilberstein, Using anytime algorithms in intelligent systems, AI Magazine 17 (3) (1996) 73–83.

Appendix A. Probability density functions for graphs

The probability density function for the second graph is given in Table A.8. This graph is borrowed from [15]. The length distribution is shown by a list of pairs (x, y), where y is the probability that the length of the given edge equals to x.

Table A.8: Length distribution of graph 2

Edge	Pairs of (lengths, probabilities)			
(1,2)	(3.0,0.3)	(5.3, 0.2)	(7.4, 0.3)	(9.4, 0.2)
(1,3)	(3.5, 0.3)	(6.2, 0.3)	(7.9, 0.2)	(8.5, 0.2)
(1,4)	(4.2, 0.2)	(6.1, 0.3)	(6.9, 0.2)	(8.9, 0.3)
(2,5)	(2.6, 0.2)	(4.1, 0.2)	(5.5, 0.4)	(9.0, 0.2)
(2,6)	(5.8, 0.3)	(7.0, 0.3)	(8.5, 0.2)	(9.6, 0.2)
(3,2)	(1.5, 0.2)	(2.3, 0.2)	(3.6, 0.3)	(4.5, 0.3)
(3,7)	(6.5, 0.5)	(7.2, 0.2)	(8.3, 0.2)	(9.4, 0.1)
(3,8)	(5.9, 0.4)	(7.8, 0.3)	(8.6, 0.1)	(9.9, 0.2)
(4,3)	(2.1, 0.2)	(3.2, 0.2)	(4.5, 0.3)	(6.8, 0.3)
(4,9)	(1.1, 0.2)	(2.2, 0.3)	(3.5, 0.4)	(4.3, 0.1)
(5,7)	(3.2, 0.2)	(4.8, 0.2)	(6.7, 0.3)	(8.2, 0.3)
(5,10)	(6.3, 0.2)	(7.8, 0.2)	(8.4, 0.4)	(9.1, 0.2)
(6,3)	(6.8, 0.4)	(7.7, 0.1)	(8.5, 0.1)	(9.6, 0.4)
(6,5)	(0.6, 0.2)	(1.5, 0.2)	(3.9, 0.3)	(5.8, 0.3)
(6,7)	(2.1, 0.2)	(4.8, 0.4)	(6.6, 0.2)	(7.5, 0.2)
(7,6)	(4.1, 0.2)	(6.3, 0.3)	(8.5, 0.4)	(9.7, 0.1)
(7,8)	(1.6, 0.2)	(2.8, 0.3)	(5.2, 0.3)	(6.0, 0.2)
(7,10)	(1.6, 0.2)	(3.4, 0.3)	(8.2, 0.3)	(9.3, 0.2)
(8,4)	(7.0, 0.2)	(8.0, 0.2)	(8.8, 0.2)	(9.4, 0.4)
(8,7)	(2.1, 0.4)	(4.6, 0.2)	(8.5, 0.2)	(9.6, 0.2)
(8,9)	(1.7, 0.2)	(4.9, 0.2)	(6.5, 0.2)	(7.8, 0.4)
(7,9)	(3.5, 0.1)	(4.0, 0.2)	(5.0, 0.4)	(7.7, 03)
(9,10)	(4.6, 0.4)	(6.4, 0.1)	(7.6, 0.2)	(8.9, 0.3)

The probability density function for the second graph is given in Table A.9. This graph is borrowed from [15]. The length distribution is shown by a list of pairs (x, y), where y is the probability that the length of the given edge equals to x.

Table A.9: Length distribution of graph 3

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