

Learning Automata-based Algorithms for Solving Stochastic Minimum Spanning Tree Problem

Javad Akbari Torkestani

Department of Computer Engineering, Islamic Azad University, Arak Branch, Arak, Iran
j-akbari@iau-arak.ac.ir

Mohammad Reza Meybodi

Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, Iran
Institute for Studies in Theoretical Physics and Mathematics (IPM), School of Computer Science, Tehran, Iran
mmeybodi@aut.ac.ir

Abstract

Due to the hardness of solving the minimum spanning tree (MST) problem in stochastic environments, the stochastic MST (SMST) problem has not received the attention it merits, specifically when the probability distribution function (PDF) of the edge weight is not a priori known. In this paper, we first propose a learning automata-based sampling algorithm (Algorithm 1) to solve the MST problem in stochastic graphs where the PDF of the edge weight is assumed to be unknown. At each stage of the proposed algorithm, a set of learning automata is randomly activated and determines the graph edges that must be sampled in that stage. As the proposed algorithm proceeds, the sampling process focuses on the spanning tree with the minimum expected weight. Therefore, the proposed sampling method is capable of decreasing the rate of unnecessary samplings and shortening the time required for finding the SMST. The convergence of this algorithm is theoretically proved and it is shown that by a proper choice of the learning rate the spanning tree with the minimum expected weight can be found with a probability close enough to unity. Numerical results show that Algorithm 1 outperforms the standard sampling method. Selecting a proper learning rate is the most challenging issue in learning automata theory by which a good trade off can be achieved between the cost and efficiency of algorithm. To improve the efficiency (i.e., the convergence speed and convergence rate) of Algorithm 1, we also propose four methods to adjust the learning rate in Algorithm 1 and the resultant algorithms are called as Algorithm 2 through Algorithm 5. In these algorithms, the probabilistic distribution parameters of the edge weight are taken into consideration for adjusting the learning rate. Simulation experiments show the superiority of Algorithm 5 over the others. To show the efficiency of Algorithm 5, its results are compared with those of the multiple edge sensitivity method (MESM). The obtained results show that Algorithm 5 performs better than MESM both in terms of the running time and sampling rate.

Keyword: Learning automata, Minimum spanning tree, Stochastic graph

1. Introduction

A minimum spanning tree (MST) of an edge-weighted network is a spanning tree having the minimum sum of edge weights among all spanning trees. The weight associated with the graph edge represents its cost, traversal time, or length depending on the context. Minimum spanning tree has many applications in different areas such as data storage [31, 32], statistical cluster analysis [33, 34], picture processing [35], and especially in communication networks [36, 37]. In network routing protocols, the minimum cost spanning tree is one of the most effective methods to multicast or broadcast the messages from a source node to a set of destinations. In most scenarios, the edge weight is assumed to be fixed, but this does not hold always true and they vary with time in real applications. For example, the links in a communication network may be affected by collisions, congestions and interferences. Therefore, the MST problem is generalized toward a stochastic minimum spanning tree (SMST) problem in which the edge weight is a random variable rather than a constant value. There have been many studies of the minimum spanning tree problem dealing with the deterministic graphs and several well-known sequential algorithms such as Boruvka [1], Kruskal [2] and Prim [3], in which the MST problem can be solved in polynomial time, have been presented. Furthermore, several efficient

distributed [20-24] and approximation [25-27] algorithms have been also proposed to solve the MST problem in deterministic graphs. However, when the edge weight is allowed to be random (vary with time), this problem becomes considerably hard to solve. This becomes more difficult if the probability distribution function (PDF) of the edge weight is not a priori known. Unlike the deterministic graphs, due to hardness the MST problem has not received much attention in stochastic graphs.

Ishii et al. [4] proposed a method for solving the stochastic spanning tree problem in which the mentioned problem is transformed into its proxy deterministic equivalent problem and then a polynomial time algorithm presented to solve the latter problem. In this method, the probability distribution of the edges' weights is assumed to be known. Ishii and Nishida [5] considered a stochastic version of the bottleneck spanning tree problem on the edges whose weights are random variables. They showed that, under reasonable restrictions, the problem can be reduced to a minimum bottleneck spanning tree problem in a deterministic case. Mohd [6] proposed a method for stochastic spanning tree problem called interval elimination and introduced several modifications to the algorithm of Ishii et al. [4] and showed that the modified algorithm is able to obtain the better results in less time. Ishii and Matsutomi [7] presented a polynomial time algorithm to solve the problem stated in [4] when the probability distribution of the edges weight is unknown. They applied a statistical approach to a stochastic spanning tree problem and considered a minimax model of the stochastic spanning tree problems with a confidence region of unknown distribution parameters. Alexopoulos and Jacobson [8] proposed some methods to determine the probability distribution of the weight of the MST in stochastic graphs in which the edges have independent discrete random weights, and the probability that a given edge belongs to a minimum spanning tree. Dhamdhare et al. [9] and Swamy and Shmoys [10] formulated the stochastic MST problem as a stochastic optimization problem and proposed some approximation approaches to solve two and multistage stochastic optimization problems.

In [41], Hutson and Shier studied several approaches to find (or to optimize) the minimum spanning tree when the edges undergo the weight changes. Repeated Prim method, cut-set method, cycle tracing method, and multiple edge sensitivity method are the proposed approaches to find the MST of the stochastic graph in which the edge weight can assume a finite number of distinct values as a random variable. To approximate the expected weight of the optimal spanning tree, Hutson and Shier used the algebraic structure to describe the relationship between the different edge-weight realizations of the stochastic graph. They compared different approaches and showed that the multiple edge sensitivity method, which is here referred to as MESM, outperforms the others in terms of the time complexity and the size of the constructed state space. Katagiri et al. [38] introduced a fuzzy-based approach to model the minimum spanning tree problem in case of fuzzy random weights. They examined the case where the edge weights are fuzzy random variables. Fangguo and Huan [39] considered the problem of minimum spanning tree in uncertain networks in which the edge weights are random variables. They initially define the concept of the expected minimum spanning tree and mathematically model (formulate) the problem accordingly. Then, based on the resulting model they propose a hybrid intelligent algorithm which is a combination of the genetic algorithm and a stochastic simulation technique to solve the SMST problem. In [44], a new classification is presented for MST algorithms. This paper generally classifies the existing approaches into deterministic and stochastic MST algorithms. Deterministic algorithms are further subdivided into constrained and unconstrained classes. In this paper, the authors propose a heuristic method to solve the stochastic version of the minimum spanning tree problem. In [39], the Prüfer encoding scheme which is able to represent all possible trees is used to code the corresponding spanning trees in the genetic representation. Almeida et al. [40] studied the minimum spanning tree problem with fuzzy parameters and proposed an exact algorithm to solve this problem.

The main drawback of the above mentioned SMST algorithms is that they are feasible only when the probability distribution function of the edge weight is assumed to be a priori known, whereas this assumption can not hold true in realistic applications. In this paper, we first propose a learning automata-based approximation algorithm called Algorithm 1 to solve the minimum spanning tree problem in a stochastic graph, where the probability distribution function of the edge weight is unknown. As Algorithm 1 proceeds, the process of sampling from the graph is concentrated on the edges of the spanning tree with the minimum expected weight. This is theoretically proved for Algorithm 1 that by the proper choice of the learning rate the spanning tree with the minimum expected weight will be found with a probability close enough to unity. To show the performance of Algorithm 1, the number of samples that is required to be taken from the stochastic graph by this algorithm is compared with that of the standard sampling method when the approximated value of the edge weight converges to its mean value with a certain probability. The obtained results show that the average number of samples taken by Algorithm 1 is much less than that of the standard sampling method. In

Algorithm 1, a learning automaton is assigned to each graph node and the incident edges of node are defined as its action-set. This algorithm is composed of a number of stages and at each stage a spanning tree of the graph is constructed by the random selection of the graph edges by the activated automata. At each stage, if the expected weight of the constructed spanning tree is larger than the average weight of all spanning trees constructed so far, the constructed spanning tree is penalized and it is rewarded otherwise. The performance of a learning automata-based algorithm is directly affected by the learning rate. Choosing the same learning rate for all learning automata to update their action probability vectors may prolong the convergence for some learning automata and accelerate the convergence to a non-optimal solution for some others. To avoid this, we also propose four methods for adjusting the learning rate and apply them to Algorithm 1. Algorithm 1 in which the learning rate is computed by these four statistical methods are named as Algorithm 2 through Algorithm 5. Computer simulation shows that Algorithm 5 outperforms the others in terms of sampling rate (the number of samples) and convergence rate. To investigate the efficiency of the best proposed algorithm (Algorithm 5), we compare Algorithm 5 with MESM proposed by Hutson and Shier [41]. The obtained results show that Algorithm 5 is superior to MESM in terms of running time and sampling rate.

The rest of the paper is organized as follows. The stochastic graph and learning automata theory are briefly reviewed in section 2. In section 3, our learning automata-based algorithms are presented. Section 4 presents the convergence proof of the first proposed algorithm (Algorithm 1) and studies the relationship between the learning rate and convergence rate of Algorithm 1. Section 5 evaluates the performance of the proposed algorithms through simulation experiments, and Section 6 concludes the paper.

2. Stochastic Graph, Learning Automata Theory

To provide the sufficient background for understanding the basic concepts of the Stochastic minimum spanning tree algorithms which are presented in this paper, the stochastic graphs and learning automata theory are briefly described in this section.

2.1. Stochastic Graph

An undirected stochastic graph G can be defined by a triple $G = (V, E, W)$, where $V = \{v_1, v_2, \dots, v_n\}$ is the vertex set, $E \subset V \times V$ is the edge set, and matrix $W_{n \times n}$ denotes the probability distribution function of the random weight associated with the graph edges, where n is the number of nodes. For each edge $e_{(i,j)}$ the associated weight is assumed to be a positive random variable with probability density function $w_{i,j}$, which is assumed to be unknown in this paper. An undirected stochastic sub-graph $G' = (V', E', W)$ is also called a stochastic spanning tree of G if G' is a connected sub-graph, G' has the same edge set as G , and $|E'| \leq |E|$, satisfying $|E'| = n-1$, where $|E'|$ denotes the cardinality of set E' . Let $T = \{\tau_1, \tau_2, \tau_3, \dots\}$ be the set of all possible stochastic spanning trees of graph G and $\bar{W}\tau_i$ denotes the expected weight of spanning tree τ_i . SMST is defined as the stochastic spanning tree with the minimum expected weight. In other words, a given stochastic minimum spanning tree $\tau^* \in T$ is the SMST if and only if $\bar{W}\tau^* = \min_{\forall \tau_i \in T} \{\bar{W}\tau_i\}$ [4, 7].

2.2. Learning Automaton

A learning automaton [11, 12] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. Learning automaton has shown to perform well in computer networks [43, 46, 48, 49, 50, 51, 52] and solving combinatorial optimization problems [44, 45, 47, 53]. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized.

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, \dots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, \dots, c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i . If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary

environment. The environments depending on the nature of the reinforcement signal β can be classified into P -model, Q -model and S -model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P -model environments. Another class of the environment allows a finite number of the values in the interval $[0, 1]$ can be taken by the reinforcement signal. Such an environment is referred to as Q -model environment. In S -model environments, the reinforcement signal lies in the interval $[a, b]$.

Learning automata can be classified into two main families [11, 13-19]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \beta\alpha, T \rangle$, where β is the set of inputs, α is the set of actions, and T is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha_i(k) \in \alpha$ and $p_j(k)$ denote the action selected by learning automaton and the probability vector defined over the action set at instant k , respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by learning automaton. At each instant k , the action probability vector $p(k)$ is updated by the linear learning algorithm given in Equation (1), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in Equation (2) if the taken action is penalized.

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

$$p_j(k+1) = \begin{cases} (1 - b)p_j(k) & j = i \\ \left(\frac{b}{r-1}\right) + (1 - b)p_j(k) & \forall j \neq i \end{cases} \quad (2)$$

If $a = b$, the recurrence equations (1) and (2) are called linear reward-penalty (L_{R-P}) algorithm, if $a \gg b$ the given equations are called linear reward- ϵ penalty ($L_{R-\epsilon P}$), and finally if $b = 0$ they are called linear reward-Inaction (L_{R-I}). In the latter case, the action probability vectors remain unchanged when the taken action is penalized by the environment.

2.3. Variable Action Set Learning Automaton

A variable action-set learning automaton is an automaton in which the number of actions available at each instant changes with time. It has been shown in [11] that a learning automaton with a changing number of actions is absolutely expedient and also ϵ -optimal, when the reinforcement scheme is L_{R-I} . Such an automaton has a finite set of n actions, $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$. $A = \{A_1, A_2, \dots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k . The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\Psi(k) = \{\Psi_1(k), \Psi_2(k), \dots, \Psi_m(k)\}$ defined over the possible subsets of the actions, where $\Psi_i(k) = \text{prob}[A(k) = A_i | A_i \in A, 1 \leq i \leq 2^n - 1]$.

$\hat{p}_i(k) = \text{prob}[\alpha(k) = \alpha_i | A(k), \alpha_i \in A(k)]$ denotes the probability of choosing action α_i , conditioned on the event that the action subset $A(k)$ has already been selected and $\alpha_i \in A(k)$ too. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = \frac{p_i(k)}{K(k)} \quad (3)$$

where $K(k) = \sum_{\alpha_i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$, and $p_i(k) = \text{prob}[\alpha(k) = \alpha_i]$.

The procedure of choosing an action and updating the action probabilities in a variable action-set learning automaton can be described as follows. Let $A(k)$ be the action subset selected at instant n . Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in Equation (3). The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the action probability vector of the chosen subset is rescaled as $p_i(k+1) = \hat{p}_i(k+1) \cdot K(k)$, for all $\alpha_i \in A(k)$. The absolute expediency and ϵ -optimality of the method described above have been proved in [11].

3. Learning Automata-based SMST Algorithms

Let $G = (V, E, W)$ denotes the input undirected stochastic graph, where V is the vertex set, E is the edge set, and matrix W denotes the probability distribution function of the random weight associated with the graph edges. A network of learning automata isomorphic to the stochastic graph is formed by assigning to each node of the graph a learning automaton. The resulting network can be described by a triple $\langle \underline{A}, \underline{\alpha}, W \rangle$, where $\underline{A} = \{A_1, A_2, \dots, A_n\}$ denotes the set of the learning automata, $\underline{\alpha} = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ denotes the set of action-sets in which $\alpha_i = \{\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{ij}, \dots, \alpha_{ir}\}$ defines the set of actions that can be taken by learning automata A_i for each $\alpha_i \in \underline{\alpha}$. Since the stochastic graph is undirected, edge $e_{(i,j)}$ corresponds either to the action α_{ij} of the learning automata A_i or to the action α_{ji} of the learning automata A_j . Weight $w_{i,j}$ denotes the random weight associated with edge $e_{(i,j)}$ (i.e., $W(e_{(i,j)})$). In this paper, it is assumed that $W(e_{(i,j)})$ be a positive random variable with an unknown probability distribution. In this algorithm which is hereafter referred to as Algorithm 1, each learning automaton can be in one of two states: active and passive. Each learning automaton is initially set to a passive state. The proposed algorithm consists of a number of stages. Stage k of the proposed algorithm is briefly described in the following steps.

1. SPANNING TREE FORMATION

Repeat the following until either the number of the selected edges is greater than or equal to $(n-1)$ or there are no more passive automata to be activated

- Choose one of the passive automata at random. Mark it as active.
- The activated automaton chooses one of its actions according to its action probability vector.
- The random weight associated with the selected action is added to the weight of the spanning tree that is being formed in the current stage.
- Each passive learning automaton changes its number of actions (or scales its action probability vector) by disabling the actions that may cause cycle during the formation of the spanning tree (The process of disabling the actions is described later).

2. DYNAMIC THRESHOLD COMPUTATION

Let us assume that spanning tree τ_i is selected at stage k . The average weight of all the constructed spanning trees until stage k is computed and compared with dynamic threshold T_k . At each stage $k > 1$, dynamic threshold T_k is computed as

$$T_k = \frac{1}{r} \sum_{i=1}^r \overline{W}(\tau_i) \quad (4)$$

where r is the number of spanning trees constructed so far, and $\overline{W}(\tau_i)$ denotes the average weight of spanning tree τ_i which is defined as

$$\overline{W}(\tau_i) = \frac{1}{k_i} \sum_{j=1}^{k_i} W\tau_i(j) \quad (5)$$

where k_i denotes the number of times spanning tree τ_i has been constructed, and $W\tau_i(k)$ denotes the weight of spanning tree τ_i at stage k which is defined as follows

$$W\tau_i(k) = \sum_{\forall e_{(s,t)} \in \tau_i} W(e_{(s,t)}(k)) \quad (6)$$

where $W(e_{(s,t)})$ is a positive random variable with unknown probability distribution and $W(e_{(s,t)}(k))$ denotes its weight at stage k .

3. UPDATING ACTION PROBABILITY VECTOR

At each stage k , if the average weight of the constructed spanning tree (i.e., $\overline{W}(\tau_i)$) is less than or equal to dynamic threshold T_k , then the actions chosen by all activated automata are rewarded and

penalized otherwise. In this algorithm, each automaton updates its action probability vector by using a L_{R-I} reinforcement scheme. The disabled actions are enabled again and the probability vectors are updated as described in Subsection 2.3.

4. TERMINATION

Steps 1, 2, and 3 are repeated until the choice probability of a spanning tree (PMST) becomes greater than a certain threshold T_{PMST} or the stage number exceeds a pre-specified threshold T_k . PMST is defined as the product of the choice probabilities of the edges of the constructed spanning tree. The spanning tree which is constructed just before the algorithm stops is the spanning tree with the minimum expected weight among all spanning trees of the stochastic graph.

Algorithm 1 The first proposed SMST algorithm

```

01:Input: Stochastic Graph  $G = \langle V, E, F \rangle$ , Thresholds  $T_k, T_{PMST}$ 
02:Output: The Minimum Spanning Tree
03: $T_k \leftarrow 0, k \leftarrow 0$ 
04:Let  $\tau$  denotes the constructed spanning tree and  $w_\tau$  denotes its weight
05:Begin Algorithm
06: $\tau \leftarrow \varphi, w_\tau \leftarrow 0$ 
07: Assign an automaton to each node and initially set it to the passive state
08: Repeat
09:   Let  $A$  denotes the set of activated automata which is initially null
10:   While  $|\tau| < |V| - 1$  or  $A < n$  Do
11:     Select one of the passive automata at random and call it  $A_i$ 
12:     If  $|\alpha_i| \neq \emptyset$  Then
13:       Automaton  $A_i$  chooses one of its actions (say  $e_{(i,j)}$ )
14:        $A \leftarrow A + A_i$ 
15:        $\tau \leftarrow \tau + e_{(i,j)}$ 
16:        $w_\tau \leftarrow w_\tau + w(e_{(i,j)})$ 
17:       Each passive automaton prunes its action-set for cycle avoidance
18:     Else
19:        $A \leftarrow A + A_i$ 
20:     End If
21:   End While
22:   If  $|\tau| = |V| - 1$  Then
23:     Compute the average weight of the selected spanning tree as  $\bar{w}_\tau$ 
24:     If  $\bar{w}_\tau \leq T_k$  Then
25:       Reward the selected actions of activated automata
26:     Else
27:       Penalize the selected actions of the activated automata
28:     End If
29:      $T_k \leftarrow [(k-1)T_{k-1} + w_\tau]/k$ 
30:   Else
31:     Do nothing
32:   End If
33:    $k \leftarrow k + 1$ 
34:   Enable all disabled actions
35: Until  $PMST > P_{PMST}$  or  $k > T_k$ 
36:End Algorithm
```

Figure 2. The pseudo code of Algorithm 1

As mentioned earlier, at each stage of Algorithm 1 the number of actions that can be taken by a passive automaton changes (or reduces) with time in such a way that no cycle appears in the spanning tree (see Line 17 of the Algorithm 1 in Figure 2). To avoid the formation of a cycle in the spanning tree, Algorithm 1

performs as follows: Let $\pi_{i,j}$ denotes the path connecting node v_i to node v_j , and p_j^i and q_j^i denote the choice probability of edge $e_{(i,j)}$ and path $\pi_{i,j}$, respectively. Now, at each stage k , the proposed algorithm removes every edge $e_{(r,s)}$ (or $e_{(s,r)}$) and temporarily disables its corresponding action in the action-set of the passive automata A_r and A_s , if edge $e_{(i,j)}$ is chosen at stage k and both paths $\pi_{i,r}$ and $\pi_{j,s}$ has been already constructed by the activated automata. This disabling process is repeated until the stop condition of step 1 is met. By this, the cycle-freeness of the proposed MST formation algorithm is guaranteed. At each stage, Algorithm 1 updates the action probability vectors twice. The first time is when the selected actions are rewarded or penalized, and the second time is when the disabled actions are enabled again at the end of each stage. In both cases, the action probabilities are updated as described in Subsection 2.3 on variable action-set learning automata.

3.1. Improvements

In Algorithm 1, all learning automata use the same learning rate remaining unchanged during the execution of algorithm. Such a learning rate gives the equal importance to all edges of the stochastic graph to being in the minimum spanning tree. This may prolong the convergence to the optimal solution for some learning automata and accelerate the convergence to a non-optimal solution for some others. In this section, we discuss four statistical methods for adjusting the learning rate so as to improve the speed and convergence rates of Algorithm 1. These methods take into account the probability distribution parameters of the edge weight for adjusting the learning rate in the course of algorithm.

Method I The number of samples that is required to be taken from a given edge to reach an accurate estimation of the mean value of its random weight is directly proportional to the variance of the random variable associated with the edge weight. That is, the sampling rate increases and the learning rate decreases as the variance of the random weight becomes larger. Therefore, it is expected that the convergence speed of Algorithm 1 is improved if the learning rate of each graph edge is adjusted based on the variance of its random weight as given in Equation (7). In this method, the variance must be initially projected to interval (0,1) for every edge. Algorithm 1 in which the learning rate is adjusted on the basis of Method I is called Algorithm 2.

$$a_{e(i,j)}(k) = a/\sigma_{e(i,j)}(k) = a \cdot \left[\sum_{k=1}^{k_{e(i,j)}} [x_{e(i,j)}(k) - \bar{x}_{e(i,j)}(k)]^2 / (k_{e(i,j)} - 1) \right]^{-\frac{1}{2}} \quad (7)$$

where $k_{e(i,j)}$ denote the number of times edge $e_{(i,j)}$ is sampled, $x_{e(i,j)}(k)$ and $\bar{x}_{e(i,j)}(k)$ denote the weight of edge $e_{(i,j)}$ at stage k and its average weight, and a denotes the constant general learning rate.

Method II An edge with a lower average weight is more likely to be a part of the final minimum spanning tree. That is, the edges having smaller weight need a lower sampling rate than the heavy weight edges. Therefore, it is expected that the convergence rate of Algorithm 1 increases if the learning rate of every edge is adjusted by Equation (8). This equation implies that the learning rate of an edge is inversely proportional to its average weight. The edge with the lowest average weight is rewarded more than the others and so the automaton converges to it by fewer samples. Like Method I, before applying Equation (8), the mean value must be projected to interval (0,1) for every edge. Algorithm 1 in which Method II is used to adjust the learning rate is called Algorithm 3.

$$a_{e(i,j)}(k) = a/\mu_{e(i,j)}(k) = a \cdot k_{e(i,j)} / \sum_{k=1}^{k_{e(i,j)}} x_{e(i,j)}(k) \quad (8)$$

Method III As the algorithm proceeds, the choice probability of the edges of the spanning tree with the minimum expected weight is gradually increases as that of the others decreases. The growth of the choice probability of an action means that it is rewarded by the environment. In fact, the choice probability of an action represents its optimality. In Equation (9), the learning rate of an edge is directly defined on the basis of its choice probability. This equation increases the convergence speed more and more as the choice probability

increases (or as the algorithm approaches to the end). Algorithm 1 in which the learning rate is computed by Equation (9) is called Algorithm 4.

$$a_{e(i,j)}(k) = a \cdot P_{e(i,j)}(k) \quad (9)$$

Method IV Equation (8) can be used in applications that sacrifice the running time in favor of the solution optimality, while Equation (9) is appropriate for applications that give significant importance to the running time. Different combinations of the previous methods (Methods I-III) can be also used for applications in which a trade-off between the running time and solution optimality is desired. We combined Method II and Method III in order to improve both the convergence rate and convergence speed of Algorithm 1. This method is called Method IV and computes the learning rate as given in Equation (10). Algorithm 1 in which Method IV is used to adjust the learning rate is called Algorithm 5.

$$a_{e(i,j)}(k) = \frac{a \cdot P_{e(i,j)}(k)}{\mu_{e(i,j)}(k)} = a \cdot P_{e(i,j)}(k) \cdot k_{e(i,j)} \left/ \sum_{k=1}^{k_{e(i,j)}} x_{e(i,j)}(k) \right. \quad (10)$$

4. Convergence Results

In this section we prove two main results of the paper. The first result concerns the convergence of Algorithm 1 to the optimal solution when each learning automaton updates its action-set by a linear reward-inaction reinforcement scheme (Theorem 1). This result represents that by choosing a proper learning rate for Algorithm 1, the choice probability of the optimal solution converges to one as much as possible. Since Algorithm 1 is designed for stochastic environments where the environmental parameters vary over time, the method that is used to prove the convergence of the Algorithm 1 partially follows the method given in [12, 13] to analyze the behavior of the learning automata operating in non-stationary environments. The second result concerns the relationship between the convergence error parameter ε (i.e. the error parameter involves in the standard sampling method) and the learning rate of the Algorithm 1 (Theorem 3). The second result aims at determining a learning rate $a(\varepsilon)$ under which the probability of constructing the spanning tree with the minimum expected weight exceeds $1 - \varepsilon$.

Theorem 1 Let $q_i(k)$ be the probability of constructing spanning tree τ_i at stage k . If $\underline{q}(k)$ is updated according to Algorithm 1, then there exists a learning rate $a^*(\varepsilon) \in (0,1)$ (for every $\varepsilon > 0$) such that for all $a \in (0, a^*)$, we have

$$\text{Prob}[\lim_{k \rightarrow \infty} q_i(k) = 1] \geq 1 - \varepsilon$$

Proof The steps of the convergence proof are briefly outlined as follows. At first, it is proved that the penalty probability of each spanning tree converges to the constant value of the final penalty probability, if k is selected large enough. This property is shown in Lemma 1. Then, it is shown that the probability of choosing the spanning tree with the minimum expected weight is a sub-Martingale process for large values of k , and so the changes in the probability of constructing the minimum spanning tree is always nonnegative. Lemmas 2 and 3 show this result. Finally, the convergence of Algorithm 1 to the spanning tree with the minimum expected weight is proved by using Martingale convergence theorems. Therefore, the following lemmas need to be proved before stating the proof of the Theorem 1.

Lemma 1 If spanning tree τ_i is penalized with probability $c_i(k)$ at stage k (i.e. $c_i(k) = \text{prob}[\bar{W}\tau_i > T_k]$), and $\lim_{k \rightarrow \infty} c_i(k) = c_i^*$. Then, for every $\varepsilon \in (0,1)$ and $k > K(\varepsilon)$ we have,

$$\text{prob}[|c_i^* - c_i(k)| > 0] < \varepsilon$$

Proof Let c_i^* denotes the final value of probability $c_i(k)$ when k is large enough. Using weak law of large numbers, we conclude that

$$\lim_{k \rightarrow \infty} \text{prob}[|c_i^* - c_i(k)| > \varepsilon] \rightarrow 0$$

Hence, for every $\varepsilon \in (0,1)$, there exists a $a^*(\varepsilon) \in (0,1)$ and $K(\varepsilon) < \infty$ such that for all $a < a^*$ and $k > K(\varepsilon)$ we have $\text{Prob}[|c_i^* - c_i(k)| > 0] < \varepsilon$, and the proof of the Lemma 1 is completed. \blacksquare

Lemma 2 Let $c_j(k) = \text{prob}[\bar{W}\tau_j(k+1) > T_k]$ and $d_j(k) = 1 - c_j(k)$ be the probability of penalizing and rewarding spanning tree τ_j (for all $j = 1, 2, \dots, r$) at stage k , respectively. If $\underline{q}(k)$ evolves according to Algorithm 1, then the conditional expectation of $q_i(k)$ is defined as

$$E[q_i(k+1) | q(k)] = \sum_{j=1}^r q_j(k) [c_j(k)q_i(k) + d_j \prod_{e(m,n) \in \tau_i} \delta_n^m(k)]$$

where

$$\delta_n^m(k) = \begin{cases} p_n^m(k+1) = p_n^m(k) + a(1 - p_n^m(k)) & ; \quad e_{(m,n)} \in \tau_j \\ p_n^m(k+1) = p_n^m(k) \cdot (1-a) & ; \quad e_{(m,n)} \notin \tau_j \end{cases}$$

where r denotes all constructed spanning trees.

Proof Since the reinforcement scheme that is used to update the probability vectors in Algorithm 1 is L_{R-I} , at each stage k the probability of choosing the spanning tree τ_i (i.e., $q_i(k)$), remains unchanged with probability $c_j(k)$ (for all $j = 1, 2, \dots, r$), when the selected spanning tree τ_j is penalized by the random environment. On the other hand, when the selected spanning tree τ_j is rewarded, the probability of choosing the edges of the spanning tree τ_i which are in the selected spanning tree τ_j increases by a given learning rate as that of the other edges decreases. To illustrate the proof of the lemma in more detail, we prove it for the minimum spanning tree of the graph shown in Figure 3. As shown in this figure, graph G has 4 nodes, 5 edges and 8 spanning trees as follows: $\tau_1 = \{e_1, e_2, e_3\}$, $\tau_2 = \{e_1, e_2, e_5\}$, $\tau_3 = \{e_2, e_3, e_4\}$, $\tau_4 = \{e_2, e_4, e_5\}$, $\tau_5 = \{e_1, e_3, e_5\}$, $\tau_6 = \{e_1, e_3, e_4\}$, $\tau_7 = \{e_1, e_4, e_5\}$, $\tau_8 = \{e_3, e_4, e_5\}$. It is also assumed that τ_1 is the spanning tree with the minimum expected weight of the graph given in figure 3.



Figure 3. Sample graph G and its minimum spanning tree

Let $q_i(k)$ be the choice probability of spanning tree τ_i at stage k . Therefore we have,

$$\begin{aligned} q_1(k) &= p_2^1(k)p_3^1(k)p_4^1(k) \\ q_2(k) &= p_2^1(k).p_4^1(k).p_4^3(k) \\ q_3(k) &= p_3^1(k).p_4^1(k).p_4^2(k) \\ q_4(k) &= p_4^1(k).p_4^2(k).p_4^3(k) \\ q_5(k) &= p_2^1(k).p_3^1(k).p_4^3(k) \\ q_6(k) &= p_2^1(k).p_3^1(k).p_4^2(k) \\ q_7(k) &= p_2^1(k).p_4^2(k).p_4^3(k) \\ q_8(k) &= p_3^1(k).p_4^2(k).p_4^3(k) \end{aligned}$$

where $p_j^i(k)$ denotes the probability of choosing action α_{ij} of automaton A_i at stage k . The conditional expectation of $q_1(k+1)$, assuming $q(k)$ is updated according to Algorithm 1, is defined as

$$\begin{aligned} E[q_1(k+1) | q(k)] &= \\ q_1(k) \left[c_1 q_1(k) + d_1(k) \{p_2^1(k) + a(1-p_2^1(k))\} \{p_3^1(k) + a(1-p_3^1(k))\} \{p_4^1(k) + a(1-p_4^1(k))\} \right] + \\ q_2(k) \left[c_2 q_1(k) + d_2(k) \{p_2^1(k) + a(1-p_2^1(k))\} \{p_3^1(k)(1-a)\} \{p_4^1(k) + a(1-p_4^1(k))\} \right] + \\ q_3(k) \left[c_3 q_1(k) + d_3(k) \{p_2^1(k)(1-a)\} \{p_3^1(k) + a(1-p_3^1(k))\} \{p_4^1(k) + a(1-p_4^1(k))\} \right] + \\ q_4(k) \left[c_4 q_1(k) + d_4(k) \{p_2^1(k)(1-a)\} \{p_3^1(k)(1-a)\} \{p_4^1(k) + a(1-p_4^1(k))\} \right] + \\ q_5(k) \left[c_5 q_1(k) + d_5(k) \{p_2^1(k) + a(1-p_2^1(k))\} \{p_3^1(k) + a(1-p_3^1(k))\} \{p_4^1(k)(1-a)\} \right] + \\ q_6(k) \left[c_6 q_1(k) + d_6(k) \{p_2^1(k) + a(1-p_2^1(k))\} \{p_3^1(k) + a(1-p_3^1(k))\} \{p_4^1(k)(1-a)\} \right] + \\ q_7(k) \left[c_7 q_1(k) + d_7(k) \{p_2^1(k) + a(1-p_2^1(k))\} \{p_3^1(k)(1-a)\} \{p_4^1(k)(1-a)\} \right] + \\ q_8(k) \left[c_8 q_1(k) + d_8(k) \{p_2^1(k)(1-a)\} \{p_3^1(k) + a(1-p_3^1(k))\} \{p_4^1(k)(1-a)\} \right] \end{aligned}$$

After simplifying all terms in the right hand side of the equation above and some algebraic manipulations, we obtain

$$\begin{aligned} E[q_1(k+1) | q(k)] &= \sum_{j=1}^8 q_j(k) [c_j(k) q_1(k) + d_j(k) \prod_{e(m,n) \in \tau_j} \delta_n^m(k)] \\ \delta_n^m(k) &= \begin{cases} p_n^m(k+1) = p_n^m(k) + a(1-p_n^m(k)) &; e_{(m,n)} \in \tau_j \\ p_n^m(k+1) = p_n^m(k) \cdot (1-a) &; e_{(m,n)} \notin \tau_j \end{cases} \end{aligned}$$

and hence the proof of the lemma. ■

Lemma 3 The increment in the conditional expectation of $q_i(k)$ is always non-negative subject to $q(k)$ is updated according to Algorithm 1. That is, $\Delta q_i(k) > 0$.

Proof. Define

$$\Delta q_i(k) = E[q_i(k+1) | q(k)] - q_i(k).$$

From Lemma 2, we have

$$\Delta q_i(k) = E[q_i(k+1) | q(k)] - q_i(k) = \sum_{j=1}^r q_j(k) [c_j(k) q_i(k) + d_j \prod_{e(m,n) \in \tau_i} \delta_n^m(k)] - q_i(k) \quad (11)$$

where

$$\delta_n^m(k) = \begin{cases} p_n^m(k+1) = p_n^m(k) + a(1-p_n^m(k)) &; e_{(m,n)} \in \tau_j \\ p_n^m(k+1) = p_n^m(k) \cdot (1-a) &; e_{(m,n)} \notin \tau_j \end{cases}$$

where $p_n^m(k)$ is the probability of choosing edge $e_{(m,n)}$ at stage k . Since the probability with which the spanning trees are constructed, rewarded or penalized is defined as the product of the probability of choosing the edges along the spanning trees, we have

$$\begin{aligned} \Delta q_i(k) &= \sum_{j=1}^r \prod_{e(m,n) \in \tau_j} p_n^m(k) \left[\prod_{e(m,n) \in \tau_j} c_n^m(k) \cdot \prod_{e(m,n) \in \tau_i} p_n^m(k) + \prod_{e(m,n) \in \tau_j} d_n^m(k) \cdot \prod_{e(m,n) \in \tau_i} \delta_n^m(k) \right] \\ &\quad - \prod_{e(m,n) \in \tau_i} p_n^m(k) \end{aligned}$$

where $\delta_n^m(k)$ is defined as given in Equation (11) and $c_n^m(k)$ is the probability of penalizing edge $e_{(m,n)}$ at stage k and $d_n^m(k) = 1 - c_n^m(k)$. At each stage, Algorithm 1 exactly chooses $(n-1)$ edges of the stochastic graph forming one of r possible spanning trees.

$$\Delta q_i(k) = \prod_{e(m,n) \in \tau_i} E[p_n^m(k+1) | p^m(k)] - \prod_{e(m,n) \in \tau_i} p_n^m(k)$$

The equality above can be rewritten as

$$\Delta q_i(k) \geq \prod_{e(m,n) \in \tau_i} (E[p_n^m(k+1) | p^m(k)] - p_n^m(k)) = \prod_{e(m,n) \in \tau_i} \Delta p_n^m(k) \quad (12)$$

and

$$\Delta p_n^m(k) = a \cdot p_n^m(k) \sum_{s \neq n}^{r_m} p_s^m(k) \cdot (c_s^m(k) - c_n^m(k))$$

$q_i(k) \in (0,1)$ for all $\underline{q} \in S_r^0$, where $S_r = \{\underline{q}(k) : 0 \leq q_i(k) \leq 1; \sum_{i=1}^r q_i(k) = 1\}$ and S_r^0 denotes the interior of S_r .

Hence, $p_n^m(k) \in (0,1)$ for all m, n . Since edge $e_{(m,n)} \in \tau_i$ is the edge with the minimum expected weight which can be selected by automaton A_m , it is shown that $c_s^m - c_n^m > 0$ for all $s \neq n$. It follows from Lemma 1 that for large values of k , $c_s^m(k) - c_n^m(k) > 0$. Therefore, we conclude that for large values of k , the right hand side of the equation above consists of the nonnegative quantities and so we have

$$\prod_{e(m,n) \in \tau_i} a \cdot p_n^m(k) \sum_{s \neq n}^{r_m} p_s^m(k) \cdot (c_s^m(k) - c_n^m(k)) \geq 0$$

and from Equation (12), we have

$$\Delta q_i(k) \geq \prod_{e(m,n) \in \tau_i} a \cdot p_n^m(k) \sum_{s \neq n}^{r_m} p_s^m(k) \cdot (c_s^m(k) - c_n^m(k))$$

which completes the proof of this lemma. ■

Corollary 1 The set of unit vectors in $S_r - S_r^0$ forms the set of all absorbing barriers of the Markov process $\{\underline{q}(k)\}_{k \geq 1}$, where $S_r^0 = \{\underline{q}(k) : q_i(k) \in (0,1); \sum_{i=1}^r q_i(k) = 1\}$

Proof Lemma 3 implicitly proves that $\{\underline{q}(k)\}$ is a sub-Martingale. Using Martingale theorems and the fact that $\{\underline{q}(k)\}$ is a non-negative and uniformly bounded function, it is concluded that $\lim_{k \rightarrow \infty} \underline{q}_i(k)$ converges to \underline{q}^* with probability one. Hence, from Equation (11), it can be seen that $\underline{q}_i(k+1) \neq \underline{q}_i(k)$ with a nonzero probability if and only if $\underline{q}_i(k) \notin \{0,1\}$, and $\underline{q}(k+1) = \underline{q}(k)$ with probability one if and only if $\underline{q}^* \in \{0,1\}$ where $\lim_{k \rightarrow \infty} \underline{q}_i(k) = \underline{q}^*$, and hence the proof is completed. ■

Let $\Gamma_i(\underline{q})$ be the probability of convergence of Algorithm 1 to unit vector e_i with initial probability vector \underline{q} . $\Gamma_i(\underline{q})$ is defined as follows

$$\Gamma_i(\underline{q}) = \text{prob}[\underline{q}_i(\infty) = 1 | \underline{q}(0) = \underline{q}] = \text{prob}[\underline{q}^* = e_i | \underline{q}(0) = \underline{q}]$$

Let $C(S_r) : S_r \rightarrow \mathfrak{R}$ be the state space of all real-valued continuously differentiable functions with bounded derivative defined on S_r , where \mathfrak{R} is the real line. If $\psi(\cdot) \in C(S_r)$, the operator U is defined as

$$U\psi(\underline{q}) = E[\psi(\underline{q}(k+1) | \underline{q}(k) = \underline{q}]] \quad (13)$$

where $E[\cdot]$ represents the mathematical expectation.

It has been shown in [13] that operator U is linear, and it preserves the non-negative functions as the expectation of a nonnegative function remains nonnegative. In other word, $U\psi(q) \geq 0$ for all $q \in S_r$, if $\psi(q) \geq 0$. If the operator U is applied n (for all $n > 1$) times repeatedly, we have

$$U^{n-1}\psi(q) = E[\psi(q(k+1) | q(1) = q)]$$

Function $\psi(q)$ is called super-regular (sub-regular) if and only if $\psi(q) \geq U\psi(q)$ ($\psi(q) \leq U\psi(q)$), for all $q \in S_r$. It has been shown in [13] that $\Gamma_i(q)$ is the only continuous solution of $U\Gamma_i(q) = \Gamma_i(q)$ subject to the following boundary conditions.

$$\begin{aligned} \Gamma_i(e_i) &= 1 \\ \Gamma_i(e_j) &= 0 \quad ; \quad j \neq i \end{aligned}$$

Define

$$\phi_i[x, q] = \frac{e^{-xq_i/a} - 1}{e^{-x/a} - 1}$$

where $x > 0$. $\phi_i[x, q] \in C(S_r)$ satisfies the boundary conditions above.

Theorem 2 Let $\psi_i(\cdot) \in C(S_r)$ be super-regular with $\psi_i(e_i) = 1$ and $\psi_i(e_j) = 0$ for $j \neq i$, then

$$\psi_i(q) \geq \Gamma_i(q)$$

for all $q \in S_r$. If $\psi_i(\cdot) \in C(S_r)$ is sub-regular with the same boundary conditions, then

$$\psi_i(q) \leq \Gamma_i(q)$$

for all $q \in S_r$.

Proof. Theorem 2 has been proved in [12]. \blacksquare

In what follows, we show that $\phi_i[x, q]$ is a sub-regular function, and $\phi_i[x, q]$ qualifies as a lower bound on $\Gamma_i(q)$. Super and sub-regular functions are closed under addition and multiplication by a positive constant, and if $\phi(\cdot)$ is super-regular then $-\phi(\cdot)$ is sub-regular. Therefore, it follows that $\phi_i[x, q]$ is sub-regular if and only if

$$\theta_i[x, q] = e^{\frac{-xq_i}{a}}$$

is super-regular.

We now determine the conditions under which $\theta_i[x, q]$ is super-regular. From the definition of operator U given in Equation (13), we have

$$\begin{aligned} U\theta_i(x, q) &= E\left[e^{\frac{-xq_i(k+1)}{a}} | q(k) = q\right] = \left[\sum_{j=1}^r q_j d_j^* e^{\frac{-x}{a} \left[\prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \in \tau_j}} (p_n^m + a(1-p_n^m)) \right]} + \sum_{j=1}^r q_j d_j^* e^{\frac{-x}{a} \left[\prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \notin \tau_j}} (p_n^m(1-a)) \right]} \right] \\ &= \left[q_i d_i^* e^{\frac{x}{a} (q_i + a(1-q_i))} + \sum_{j \neq i} q_j d_j^* e^{\frac{-x}{a} \left[\prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \in \tau_j}} (p_n^m + a(1-p_n^m)) \right]} + \sum_{j \neq i} q_j d_j^* e^{\frac{-x}{a} \left[\prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \notin \tau_j}} (p_n^m(1-a)) \right]} \right] \end{aligned}$$

where d_j^* denotes the final value to which the reward probability $d_j(k)$ is converged (for large values of k),

and $e^{-\frac{x}{a}(q_i+a(1-q_i))}$ is the expectation of $\theta_i(x,q)$ when the minimum spanning tree τ_i is rewarded by the environment.

$$U\theta_i(x,q) = \left[q_i d_i^* e^{-\frac{x}{a}(q_i+a(1-q_i))} + \sum_{j \neq i} q_j d_j^* e^{-\frac{x}{a}(q_j+a(1-q_j))} \right] - \frac{x}{a} \left[q_i (1-a) \cdot \prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \in \tau_j}} \frac{(p_n^m + a(1-p_n^m))}{(p_n^m(1-a))} \right]$$

$$= \left[\sum_{j=i} q_j d_j^* e^{-\frac{x}{a}\rho_j^i(q_j+a(1-q_j))} + \sum_{j \neq i} q_j d_j^* e^{-\frac{x}{a}(\rho_j^i q_j(1-a))} \right]$$

where $\rho_j^i > 0$ is defined as

$$\rho_j^i = \begin{cases} \prod_{\substack{e(m,n) \in \tau_i, \\ e(m,n) \in \tau_j}} \frac{(p_n^m + a(1-p_n^m))}{(p_n^m(1-a))} & ; \quad i \neq j \\ 1 & ; \quad i = j \text{ or } (\tau_i \cap \tau_j) = \emptyset \end{cases}$$

$$U\theta_i(x,q) - \theta_i(x,q) = \left[e^{\frac{xq_i\rho_j^i}{a}} \sum_{j=i} q_j d_j^* e^{-x(1-q_j)\rho_j^i} + e^{\frac{xq_i\rho_j^i}{a}} \sum_{j \neq i} q_j d_j^* e^{xq_j\rho_j^i} \right] - e^{\frac{xq_i}{a}}$$

$\theta_i(x,q)$ is super-regular if

$$e^{\frac{xq_i\rho_j^i}{a}} \sum_{j=i} q_j d_j^* e^{-x(1-q_j)\rho_j^i} + e^{\frac{xq_i\rho_j^i}{a}} \sum_{j \neq i} q_j d_j^* e^{xq_j\rho_j^i} \leq e^{\frac{xq_i}{a}}$$

and

$$U\theta_i(x,q) \leq e^{\frac{xq_i}{a}} q_i d_i^* e^{-x(1-q_i)} + e^{\frac{xq_i}{a}} \sum_{j \neq i} q_j d_j^* e^{xq_j}$$

If $\theta_i(x,q)$ is super-regular. Therefore, we have

$$U\theta_i(x,q) - \theta_i(x,q) \leq \left[e^{\frac{xq_i}{a}} q_i d_i^* e^{-x(1-q_i)} + e^{\frac{xq_i}{a}} \sum_{j \neq i} q_j d_j^* e^{xq_j} \right] - e^{\frac{xq_i}{a}}$$

After multiplying and dividing the right hand side of the inequality above by $-xq_i$ and some algebraic simplifications, we have

$$\begin{aligned}
U\theta_i(x, q) - \theta_i(x, q) &\leq -xq_i e^{\frac{-xq_i}{a}} \left[q_i d_i^* \frac{e^{-x(1-q_i)} - 1}{-xq_i} - \sum_{j \neq i} q_j d_j^* \frac{e^{xq_j} - 1}{xq_i} \right] \\
&= -xq_i e^{\frac{-xq_i}{a}} \left[d_i^* \frac{e^{-x(1-q_i)} - 1}{-x} - \sum_{j \neq i} q_j d_j^* \frac{e^{xq_j} - 1}{xq_i} \right] \\
&= -xq_i e^{\frac{-xq_i}{a}} \left[(1-q_i) d_i^* \frac{e^{-x(1-q_i)} - 1}{-x(1-q_i)} - \sum_{j \neq i} q_j d_j^* \frac{e^{xq_j} - 1}{xq_i} \right]
\end{aligned}$$

and

$$V[u] = \begin{cases} \frac{e^u - 1}{u} &; u \neq 0 \\ 1 &; u = 0 \end{cases}$$

$$\begin{aligned}
U\theta_i(x, q) - \theta_i(x, q) &\leq -xq_i e^{\frac{-xq_i}{a}} \left[(1-q_i) d_i^* V[-x(1-q_i)] - \left(\sum_{j \neq i} q_j d_j^* V[xq_j] \right) \right] \\
&= -xq_i \theta_i(x, q) G_i(x, q)
\end{aligned}$$

where $G_i(x, q)$ is defined as

$$G_i(x, q) = (1-q_i) d_i^* V[-x(1-q_i)] - \left(\sum_{j \neq i} q_j d_j^* V[xq_j] \right) \quad (16)$$

Therefore, $\theta_i(x, q)$ is super-regular if

$$G_i(x, q) \geq 0 \quad (17)$$

for all $q \in S_r$.

From Equation (16), it follows that $\theta_i(x, q)$ is super-regular if we have

$$f_i(x, q) = \frac{V[-x(1-q_i)]}{V[xq_i]} \leq \frac{\sum_{j \neq i} q_j d_j^*}{(1-q_i) d_i^*} \quad (18)$$

The right hand side of the inequality (18) consists of the nonnegative terms, so we have

$$\left(\sum_{j \neq i} q_j \right) \min_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right) \leq \frac{1}{(1-q_i)} \sum_{j \neq i} q_j \frac{d_j^*}{d_i^*} \leq \left(\sum_{j \neq i} q_j \right) \max_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right)$$

Substituting $\sum_{j \neq i} q_j$ by $(1-q_i)$ in the above inequality, we can rewrite it as

$$\min_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right) \leq \frac{\sum_{j \neq i} q_j \frac{d_j^*}{d_i^*}}{\sum_{j \neq i} q_j} \leq \max_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right)$$

From Equation (18), it follows that $\theta_i(x, q)$ is super-regular if we have

$$f_i(x, q) \geq \max_{j \neq i} (d_j^* / d_i^*)$$

For further simplification, let employ logarithms. Let

$$\Delta(q, x) = \ln f_i(x, q)$$

It has been shown in [13] that

$$-\int_0^x H'(u)du \leq \Delta(q, x) \leq -\int_{-x}^0 H'(u)du$$

$$H(u) = \frac{dH(u)}{du}, \quad H(u) = \ln V(u)$$

Therefore, we have

$$\frac{1}{V[x]} \leq \frac{V[-x(1-q_i)]}{V[xq_i]} \leq V[-x]$$

and

$$\frac{1}{V[x]} = \max_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right) \quad (19)$$

Let x^* be the value of x for which Equation (19) is true. It is shown that there exists a value of $x > 0$ under which the Equation (19) is satisfied, if (d_j/d_i) is smaller than 1 for all $j \neq i$. By choosing $x = x^*$ Equation (19) holds true. Consequently, Equation (17) is true and $\theta_i(x, q)$ is a super-regular function. Therefore,

$$\phi_i[x, q] = \frac{e^{-xq_i/a} - 1}{e^{-x/a} - 1}$$

is a sub-regular function satisfying the boundary conditions given in Equation (14). From Theorem 2 and Inequality (15), we conclude that

$$\phi_i[x, q] \leq \Gamma_i(q) \leq 1$$

From definition of $\phi_i[x, q]$, we see that given any $\varepsilon > 0$ there exists a positive constant $a^* < 1$ such that

$$1 - \varepsilon \leq \phi_i[x, q] \leq \Gamma_i(q) \leq 1$$

for all $0 < a \leq a^*$.

Thus we conclude that the probability with which Algorithm 1 constructs the spanning tree with the minimum expected weight is equal to 1 as k converges to infinity, and so Theorem 1 is proved. ■

Theorem 3 Let $q_i(k)$ be the probability of constructing minimum spanning tree τ_i at stage k , and $(1 - \varepsilon)$ be the probability with which Algorithm 1 converges to spanning tree τ_i . If $q_i(k)$ is updated by Algorithm 1, then for every error parameter $\varepsilon \in (0, 1)$ there exists a learning rate $a \in (\varepsilon, q)$ so that

$$\frac{xa}{e^{xa} - 1} = \max_{j \neq i} \left(\frac{d_j}{d_i} \right)$$

where $1 - e^{-xq_i} = (1 - e^{-x}) \cdot (1 - \varepsilon)$ and $q_i = [q_i(k) | k = 0]$.

Proof It has been proved in [13, 17] that there always exists a $x > 0$ under which Equation (19) is satisfied, if $d_j/d_i < 1$ for all $j \neq i$. Hence, it is concluded that

$$\phi_i[x, q] \leq \Gamma_i(q) \leq \frac{1 - e^{-xq_i}}{1 - e^{-x}}$$

where q_i is the initial choice probability of the optimal spanning tree τ_i . From Theorem 1, we have for each $0 < a < a^*$ the probability of converging Algorithm 1 to the spanning tree with the minimum expected weight is $(1 - \varepsilon)$ where $a^*(\varepsilon) \in (0, 1)$. Therefore, we conclude that

$$\frac{1 - e^{-xq_i}}{1 - e^{-x}} = 1 - \varepsilon \quad (20)$$

It is shown that for every error parameter $\varepsilon \in (0,1)$ there exists a value of x under which Equation (19) is satisfied, and so we have

$$\frac{x^* a}{e^{x^* a} - 1} = \max_{j \neq i} \left(\frac{d_j^*}{d_i^*} \right)$$

It is concluded that for every error parameter $\varepsilon \in (0,1)$ there exists a learning rate $a \in (\varepsilon, q)$ under which the probability of converging Algorithm 1 to the spanning tree with the minimum expected weight is greater than $(1-\varepsilon)$ and hence the proof of the theorem. ■

5. Numerical Results

To study the efficiency of the proposed algorithms, we have conducted three set of simulation experiments on four well-known stochastic benchmark graphs borrowed from [41, 42]. The first set of experiments aims to investigate the relationship between the learning rate of Algorithm 1 and the error rate in standard sampling method. These experiments show that for every error rate ε , there exists a learning rate a^* such that for $a \leq a^*$ algorithm 1 construct the spanning tree with the minimum expected weight. This set of experiment is conducted on stochastic graphs Alex2-B and Alex3-B [41]. The second set of simulation experiments compare the performance of the SMST algorithms proposed in this paper (Algorithm 1 to Algorithm 5). These experiments are conducted on Alex2-B and Alex3-B too. The third set of experiments aims to show the efficiency of Algorithm 5 which is the best proposed algorithm in comparison with MESM (multiple edge sensitivity method)[41]. In all three set of experiments conducted in this paper, the reinforcement scheme by which the action probability vector is updated is L_{R-I} . Each algorithm stops if the probability of the constructed spanning tree (PMST) is equal to or greater than 0.95 or the number of constructed spanning trees exceeds a pre-defined threshold 100,000. For each experiment, the results are averaged over 100 different independent runs.

5.1. Experiment I

This set of experiments is conducted on two sparse graphs called Alex2 and Alex3. Alex2 comprises 9 nodes and 15 edges, and Alex3 has 10 nodes and 21 edges. The discrete random variable associated with the edge weight of Alex2 and Alex3 has two and three states in mode A and B, respectively. The probability distribution of the random weight assigned to the edges of Alex2 and Alex3 tends toward the smaller edge weights. In other words, the higher probabilities are assigned to the edges with smaller weights. Such a biased distribution is more pragmatic for modeling the network dynamics than a simple uniform distribution. The first set of experiment compares the efficiency of Algorithm 1 with the standard sampling method in terms of the sampling rate. To do so, by Theorem 3, the learning rate corresponding to each confidence level $1 - \varepsilon$ in initially computed (see Tables 1 and 2). Then, Algorithm 1 is run with the obtained learning rate and its sampling rate is measured. Finally, the sampling rate of Algorithm 1 is compared with that of the standard sampling method to obtain the same confidence level.

The number of samples that the standard sampling method takes from Alex2-B and Alex3-B to obtain a confidence level $1 - \varepsilon$ has been computed and shown in Tables 1 and 2, respectively. To do this, we varied ε from 0.01 to 0.5 and computed the minimum required number of samples for every edge of the graph in standard sampling method subject to $\text{prob}[|\bar{x}_n - \mu| < \delta] \geq 1 - \varepsilon$, where $\delta = 0.001$ [29]. According to the standard sampling method presented in [46], to obtain a confidence level $1 - \varepsilon$ for the MST, we need to build a confidence with level $1 - \varepsilon_i$ for each edge e_i such that $\sum_{i=1}^{n-1} \varepsilon_i = \varepsilon$. We assume that the edges of the stochastic graph all have the same confidence level $1 - \varepsilon_0$. Therefore, selecting $\varepsilon_0 = \varepsilon/(n-1)$, where n denotes the number of vertices of the stochastic graph, guarantees a confidence level $1 - \varepsilon$ for the MST[30, 46].

Table 1. The total number of samples taken from Alex2-B for standard sampling method

Edge	Confidence Level For MST								
	0.5	0.6	0.7	0.75	0.8	0.85	0.9	.95	.99
e ₁	299	366	323	314	282	365	555	۷۷۷	۷۷۷
e ₂	359	473	451	475	412	345	324	۴۹۹	۳۸۴
e ₃	761	600	843	694	700	665	572	۶۱۱	۷۹۷
e ₄	307	314	357	337	404	282	259	۷۲۹	۴۵۷
e ₅	295	394	289	294	373	336	265	۴۲۹	۴۵۰

e ₆	547	492	491	526	410	424	535	432	49.
e ₇	380	254	179	403	299	346	310	215	521
e ₈	331	315	319	302	403	368	514	258	551
e ₉	291	398	424	301	346	351	342	293	419
e ₁₀	686	538	360	671	604	704	667	655	712
e ₁₁	327	321	344	549	531	477	450	544	426
e ₁₂	293	276	379	274	336	447	373	407	325
e ₁₃	386	497	632	507	456	527	554	448	911
e ₁₄	463	452	686	448	562	567	550	705	501
e ₁₅	329	447	365	376	462	452	410	227	435
Total	6062	6141	6449	6479	6586	6663	6686	6884	7725

Table 2. The total number of samples taken from graph 2 for standard sampling method

Edge	Confidence Level For MST								
	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	0.99
e ₁	315	273	288	394	275	249	471	434	524
e ₂	459	473	519	397	481	552	473	491	608
e ₃	460	401	338	436	293	344	401	434	581
e ₄	346	514	351	413	395	476	401	376	579
e ₅	751	557	551	601	482	580	679	869	570
e ₆	364	304	284	268	372	320	286	452	464
e ₇	359	365	292	328	326	459	492	384	421
e ₈	279	268	240	248	319	318	353	385	512
e ₉	390	421	438	386	448	544	390	392	516
e ₁₀	304	392	467	405	352	407	428	446	588
e ₁₁	229	317	333	390	479	364	342	491	524
e ₁₂	524	410	490	580	438	428	406	550	463
e ₁₃	305	294	265	371	426	347	302	388	506
e ₁₄	400	410	443	337	377	353	434	388	500
e ₁₅	293	248	357	333	372	466	483	388	559
e ₁₆	705	507	692	474	552	482	509	555	593
e ₁₇	398	439	434	529	504	441	512	523	493
e ₁₈	327	254	342	347	340	434	444	442	593
e ₁₉	503	412	351	469	464	509	420	453	585
e ₂₀	704	765	530	706	505	556	666	536	546
e ₂₁	337	400	463	494	373	343	410	421	50..
Total	8761	8433	8478	8917	8582	8981	9310	9829	10223

Using the numerical methods, parameter x can be determined by solving equation $e^{-xq_i} + e^{-x}(1 - \varepsilon) - \varepsilon = 0$ derived from Equation (20) for every error rate ε . Then, learning rate a is calculated for the obtained parameter x by solving equation $\frac{ax}{e^{ax}-1} = \max_{j \neq i}(\frac{d_j}{d_i})$. In this experiment, we changed error rate ε from 0.5 to 0.01 (or convergence rate from 50% to 99%) and computed the corresponding learning rate for Alex2-B and Alex3-B. Then, we ran Algorithm 1 over these two benchmark graphs and measured the sampling rate of algorithm. The obtained results are shown in Tables 3 and 4. These tables also include the sampling rate of the standard sampling method to achieve the same confidence level taken from Tables 1 and 2. From the results given in Tables 3 and 4, it can be seen that the sampling rate of Algorithm 1 is much less than that of the standard sampling method for the same confidence level $(1 - \varepsilon)$ in both Alex2-B and Alex3-B.

Table 3. Sampling rate of Algorithm 1 and the standard sampling method for Alex2-B

Convergence Rate	Sampling Rate	Sampling Rate	
		Algorithm 1	Standard Sampling Method
0.50	0.1300	79	6062
0.60	0.1051	93	6141
0.70	0.0800	161	6449
0.75	0.0680	252	6479
0.80	0.0580	332	6586
0.85	0.0500	426	6663
0.90	0.0340	664	6686
0.95	0.0207	1391	6884
0.99	0.0106	3658	7455

Table 4. Sampling rate of Algorithm 1 and the standard sampling method for Alex3-B

Convergence Rate	Learning rate	Sampling Rate	
		Algorithm 1	Standard Sampling Method
0.50	0.0945	152	8761
0.60	0.0710	281	8433
0.70	0.0592	492	8478
0.75	0.0440	672	8917
0.80	0.0412	741	8582
0.85	0.0379	1391	8981
0.90	0.0256	2442	9310
0.95	0.0183	3902	9829

5.2. Experiment II

This set of experiments is conducted to compare the performance of the learning automata-based SMST algorithms presented in this paper. In these experiments, the learning rate changes from 0.005 to 0.07 and each algorithm is executed on Alex2-B and Alex3-B. The obtained results are shown in Tables 5 and 6. In these tables, for each algorithm the reported results include the total number of the samples taken from the graph (ST), the total number of the samples taken from the spanning tree with the minimum expected weight (MST), and the percentage of the converged runs (i.e., the percentage of runs converged to the minimum spanning tree) (PC).

From the results of Experiment II, the following points can be made. The experiments show that for all proposed algorithms the total number of converged runs increases and the sampling rate increase as the learning rate of algorithm decreases. For example, the convergence rate is 78% and the number of samples is 255 if Algorithm 1 is executed on Alex2-B with learning rate 0.07, whereas they are 98% and 4405 when the learning rate increases to 0.009 (see Table 3).

Comparing Algorithm 1 and Algorithm 2, we observe that the sampling rate and the convergence rate of Algorithm 2 are much less than those of Algorithm 1. The reason for such a reduction in sampling rate of Algorithm 2 is the fact that in this algorithm the learning rate of an edge is adjusted based on the variance of its random weight as given in Equation (7). That is, the number of samples that Algorithm 2 takes from the edges having small variance is significantly less than that of Algorithm 1. The results also show that for the same convergence rate, the sampling rate of Algorithm 2 is lower than Algorithm 1.

From the results given in Tables 5 and 6 for Algorithm 3, it is concluded that this algorithm is capable of improving the convergence rate of Algorithm 1. This is because Algorithm 3 takes into consideration the mean of the random weight of an edge to adjust its learning rate as described in Method II. This means that the edges with smaller mean have a larger learning rate. On the other hand, it is clear that the edge with a lower average weight is more likely to be a part of the final minimum spanning tree. Therefore, the edges of the minimum spanning tree are rewarded more with a larger learning rate and so Algorithm 3 converges to the optimal solution with a higher rate than Algorithm 1.

Comparing the results of different algorithms, it can be seen that the convergence speed of Algorithm 4 is significantly higher than that of the other algorithms as expected. This reduces the sampling rate of Algorithm 4 as compared to the others. In this algorithm, the learning rate of the edge increases as its choice probability increases. Therefore, the learning rate of the edges belonging to the minimum spanning tree and so the convergence speed of algorithm (to the optimal solution) increases as algorithm proceeds.

As a combination of Algorithm 3 and Algorithm 4, Algorithm 5 inherits both higher convergence rate and convergence speed from these algorithms. From the results shown in Tables 5 and 6, it is obvious that the convergence rate of Algorithm 5 is much higher than that of Algorithm 3. The obtained results also show that for the same convergence rate the sampling rate of Algorithm 5 is lower than that of Algorithm 4.

Table 5. Comparison of the proposed algorithms on Alex2-B for different learning rates

Learning rate	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5		
	ST	MST	PC	ST	MST	PC	ST	MST	PC	ST	MST	PC	ST	MST	PC
0.005	6052	4471	100	542	239	88	37905	21871	100	2615	858	100	28103	10267	100
0.006	5821	3887	100	480	246	88	30029	16175	100	2177	665	99	23174	9027	100
0.007	5050	3104	99	455	181	87	29511	17387	100	1851	584	99	19570	7440	100
0.008	4967	3318	100	347	159	85	26670	15982	100	1577	491	98	17291	6050	100
0.009	4405	2372	98	319	149	86	21589	14043	100	1415	426	97	15627	5468	100
0.01	3337	1889	97	266	125	85	17881	11520	100	1338	414	97	14179	5185	100
0.02	1250	697	94	138	69	83	9505	6227	100	774	296	96	7508	3152	100
0.03	921	439	91	82	44	81	5261	3323	100	455	167	96	4930	2070	100
0.04	489	272	84	63	35	78	4760	3118	100	342	120	95	4132	1915	100
0.05	395	249	83	56	26	71	3471	2326	98	293	94	92	3295	1550	100
0.06	267	143	79	45	23	68	3173	2090	99	253	89	93	2733	1277	100
0.07	255	127	78	44	17	70	2394	1529	97	209	84	91	2332	1106	100

Table 6. Comparison of the proposed algorithms on Alex3-B for different learning rates

Learning rate	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5		
	ST	MST	PC	ST	MST	PC	ST	MST	PC	ST	MST	PC	ST	MST	PC
0.004	11001	6410	100	1905	861	89	43721	28090	100	3956	838	100	40951	20255	100
0.005	8583	4560	100	1461	637	88	42422	27869	100	3298	694	99	32734	17378	100
0.006	7693	3706	98	1093	454	87	36816	25758	100	2637	592	98	27052	15038	100
0.007	7041	2931	99	887	374	88	35016	25886	100	2273	505	98	23499	13046	100
0.008	6276	3814	97	908	371	85	31200	23409	100	2164	459	98	20154	10952	100

0.009	5402	3176	95	641	238	82	23663	15983	100	1823	415	96	17951	9983	100
0.01	5258	2855	94	511	237	76	20664	13579	100	1615	318	97	16191	9014	100
0.02	3203	1329	90	310	137	69	13332	9769	100	865	219	96	8010	4556	100
0.03	1148	548	86	157	79	62	7376	5293	100	598	156	94	5691	3298	100
0.04	857	422	80	141	81	59	5276	3798	100	469	118	92	4293	2533	100
0.05	532	261	72	107	52	56	4839	3525	99	369	97	90	3397	2011	100
0.06	400	178	68	98	49	52	3259	2292	100	303	79	88	2890	1677	100
0.07	278	140	67	65	41	55	3064	2222	98	261	71	88	2461	1432	100

5.3. Experiment III

From the results of Experiment II, we observe that Algorithm 5 considerably outperforms the others both in terms of the convergence rate and sampling rate. The aim of the third set of experiments (Experiment III) is to investigate the performance of the best proposed algorithm (Algorithm 5) as opposed to multiple edge sensitivity method (MESM) proposed by Hutson and Shier [41]. In Experiment III, in addition to the sparse graphs Alex2 and Alex3, all algorithms are also tested on two complete graphs with 5 and 6 vertices called K_5 and K_6 [41, 42]. The edge weight distribution of complete graphs K_5 and K_6 can take 4 and 3 states, respectively. The probability distribution functions of K_5 and K_6 are defined in two different modes E_1 and E_2 . In mode E_1 , the variance of the random edge weight is small, while in mode E_2 it is large. The learning rate of Algorithm 5 is set to 0.07 by which this algorithm always converges to the spanning tree with the minimum expected weight. In this experiment, the metrics of interest are the average running time of algorithm (in second) that is shown in Table 7, and the sampling rate of algorithm (i.e., the total number of samples that must be taken from the graph) which is shown in Table 8.

Table 7. The average running time of algorithms (in second)

Graph	Vertices	Edges	MESM	Algorithm 5
Alex2-A	9	15	7.231	1.912
Alex2-B	9	15	28.12	2.821
Alex3-A	10	21	14.01	2.015
Alex3-B	10	21	31.14	3.480
K_5 -E1	5	10	8.450	2.211
K_5 -E2	5	10	12.98	3.665
K_6 -E1	6	15	53.87	7.010
K_6 -E2	6	15	69.44	9.150

Table 7 shows the average running time of Algorithm 5 and MESM for benchmarks Alex2, Alex3, K_5 and K_6 in second. From the results, it can be seen that K_5 and K_6 are more time consuming in contrast with Alex2 and Alex3. This is because of the larger state space of the complete graphs K_5 and K_6 as compared to Alex2 and Alex3. In fact, to find the optimal solution in a larger state space, a larger number of samples must be taken. Comparing the results of Algorithm 5 and MESM given in Table 7, it is obvious that the running time of Algorithm 5 is meaningfully shorter than that of MESM, specifically for complete graphs. It must be noted that by choosing learning rate 0.07 the percentage of the converged runs is 100 for Algorithm 5, while the convergence rate of MESM to the optimal solution is at most 90 percent. However, the running time of MESM is very longer than that of Algorithm 5. This is because MESM constructs the large state space of the problem and this in itself takes a long time for moderate size or even small stochastic graphs. Furthermore, finding the optimal state from an extremely large state space requires a lot of time. While, Algorithm 5 only constructs the parts of the problem state space in which the probability of finding the optimal or near optimal solutions is higher. Besides, as Algorithm 5 proceeds, the constructed problem state space becomes finer and finer as the choice probability of the non-optimal solutions converges to zero.

Table 8. The average sampling rate of different algorithms

Graph	Vertices	Edges	MESM	Algorithm 5
Alex2-A	9	15	23412	1814
Alex2-B	9	15	80981	2332
Alex3-A	10	21	36915	2045
Alex3-B	10	21	90440	2461
K_5 -E1	5	10	53902	3110
K_5 -E2	5	10	62411	4137
K_6 -E1	6	15	60787	7045
K_6 -E2	6	15	71192	8972

Table 8 includes the results of the simulation experiments conducted to measure the sampling rate of Algorithm 5 and MESM. From the obtained results, we observe that the sampling rate of Algorithm 5 is much less than that of MESM. This is because Algorithm 5 focuses on the sampling from the edges of the optimum spanning tree (avoids unnecessary samples) by excluding the non-optimal edges (the edges that do not belong to the optimal spanning tree) from the sampling process at each stage. In MESM, sampling from the very huge state space of the stochastic problem results in a much higher sampling rate compared to Algorithm 5. Briefly speaking, the higher convergence rate and lower sampling rate of Algorithm 5 confirm its superiority over MESM.

6. Conclusion

In this paper, we first presented a learning automata-based algorithm called Algorithm 1 for finding a near optimal solution to the MST problem in stochastic graphs where the probability distribution function of the edge weight is unknown. The convergence of the proposed algorithm was theoretically proved. Convergence results confirmed that by a proper choice of the learning rate the probability of choosing the spanning tree with the minimum expected weight converges to one. Algorithm 1 was compared with the standard sampling method in terms of the sampling rate and the obtained results showed that this algorithm outperforms it. Then, to improve the convergence rate and convergence speed of Algorithm 1, we proposed four methods to adjust the learning rate of this algorithm. The algorithms in which the learning rate is determined by these four statistical methods were called as Algorithm 2 through Algorithm 5. Simulation experiments showed that Algorithm 5 performs better than the other algorithms in terms of the sampling and convergence rates and Algorithm 1 has the worst results. To show the performance of the best proposed SMST algorithm (Algorithm 5), we compared its results with those of the multiple edge sensitivity method (MESM) proposed by Hutson and Shier [41]. Experimental results confirmed the superiority of Algorithm 5 over MESM both in terms of the running time and sampling rate.

References

- [1] O. Boruvka, "O jistem Problemu Minimalním (about a certain minimal problem)," *Práce Moravské Prirodovedecké Společnosti*, 1926, Vol. 3, pp. 37–58.
- [2] J. B. Kruskal, "On the Shortest Spanning Sub Tree of a Graph and the Traveling Salesman Problem," *Proc. Amer. Math. Soc.*, 1956, Vol. pp. 748–750.
- [3] R. C. Prim, "Shortest Connection Networks and Some Generalizations," *Bell. Syst. Tech. Journal*, 1957, Vol. 36, pp. 1389–1401.
- [4] H. Ishii, S. Shiode, T. Nishida and Y. Namasuya, "Stochastic Spanning Tree Problem," *Discrete Applied Mathematics*, 1981, Vol. 3, pp. 263–273.
- [5] H. Ishii and T. Nishida, "Stochastic Bottleneck Spanning Tree Problem," *Networks*, 1983, Vol. 13, pp. 443–449.
- [6] I. B. Mohd, "Interval Elimination Method for Stochastic Spanning Tree Problem," *Applied Mathematics and Computation*, 1994, Vol. 66, pp. 325–341.
- [7] H. Ishii and T. Matsutomi, "Confidence Regional Method of Stochastic Spanning Tree Problem," *Mathl. Comput. Modelling* Vol. 22, No. 19–22, 1995, pp. 77–82.
- [8] C. Alexopoulos and J. A. Jacobson, "State Space Partition Algorithms for Stochastic Systems with Applications to Minimum Spanning Trees," *Networks*, Vol. 35, No. 2, 2000, pp. 118–138.
- [9] K. Dhamdhere, R. Ravi and M. Singh, "On Two-Stage Stochastic Minimum Spanning Trees," *Springer-Verlag Berlin*, 2005, pp. 321–334.

- [10] C. Swamy and D. B. Shmoys, " Algorithms Column: Approximation Algorithms for 2-Stage Stochastic Optimization Problems," *ACM SIGACT News*, 2006, Vol. 37, No. 1, pp. 1-16.
- [11] M. A. L. Thathachar and B. R. Harita, "Learning Automata with Changing Number of Actions," *IEEE Transactions on Systems, Man, and Cybernetics*, 1987, Vol. SMG17, pp. 1095-1100.
- [12] K. S. Narendra and K. S. Thathachar, "Learning Automata: An Introduction", *New York, Printice-Hall*, 1989.
- [13] S. Lakshmivarahan and M. A. L. Thathachar, "Bounds on the Convergence Probabilities of Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, 1976, Vol. SMC-6, pp. 756-763.
- [14] J. Zhou, E. Billard and S. Lakshmivarahan, " Learning in Multilevel Games with Incomplete Information-Part II," *IEEE Transactions on Syystem, Man, and, Cybernetics*, 1999, Vol. 29, No.3, pp. 340-349.
- [15] M. A. L. Thathachar and P. S. Sastry, "A Hierarchical System of Learning Automata That Can Learn the Globally Optimal Path," *Information Science*, 1997, Vol.42, pp.743-766.
- [16] M. A. L. Thathachar, V. V. Phansalkar, "Convergence of Teams and Hierarchies of Learning Automata in Connectionist Systems," *IEEE Transactions on Systems, Man and Cybernetics*, 1995, Vol. 24, pp. 1459-1469.
- [17] S. Lakshmivarahan and K. S. Narendra, "Learning Algorithms for Two Person Zero-sum Stochastic Games with Incomplete Information: A unified approach," *SIAM Journal of Control Optim.*, 1982, Vol. 20, pp. 541-552.
- [18] K. S. Narenrdra and M. A. L. Thathachar, "On the Behavior of a Learning Automaton in a Changing Environment with Application to Telephone Traffic Routing," *IEEE Transactions on Systems, Man, and, Cybernetics*,1980, Vol. SMC-10, No. 5, pp. 262-269.
- [19] H. Beigy and M. R. Meybodi, "Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problems," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2006, Vol.14, pp. 591-615.
- [20] R.G. Gallager, P.A. Humblet and P.M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Trans. on Program. Lang. & Systems*, 1983, Vol. 5, pp. 66-77.
- [21] F. Chin and H.F. Ting, "An Almost Linear Time and $O(n \log n + e)$ Messages Distributed Algorithm for Minimum-Weight Spanning Trees," in *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1985, pp. 257- 266.
- [22] E. Gafni, "Improvements in the Time Complexity of Two Message-Optimal Election Algorithms," in *Proceedings s of the 4th Symposium on Principles of Distributed Computing (PODC)*, 1985, pp. 175-185.
- [23] B. Awerbuch, "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems," in *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC)*, 1987, pp 230-240.
- [24] J. Garay, S. Kutten, and D. Peleg, "A Sublinear Time Distributed Algorithm for Minimum-Weight Spanning Trees," *SIAM Journal of Computing*, 1998, Vol. 27, pp. 302-316.
- [25] M. Elkin, "Unconditional Lower Bounds on the Time-Approximation Tradeoffs for the Distributed Minimum Spanning Tree Problem," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2004, pp. 331-340.
- [26] M. Elkin, "An Overview of Distributed Approximation," *ACM SIGACT News Distributed Computing Column*, 2004, Vol. 35 No. 4, pp. 40-57.

- [27] M. Khan and G. Pandurangan, "A Fast Distributed Approximation Algorithm for Minimum Spanning Trees," in *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, September 2006.
- [28] F. B. ALT, "Bonferroni Inequalities and Intervals," in *Encyclopedia of Statistical Sciences*, 1982, Vol. 1, pp. 294–301.
- [29] C. E. Bonferroni, "Teoria Statistica Delle Classi e Calcolo Delle Probabilità," *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 1936, Vol. 8, pp. 3–62.
- [30] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," *Scandinavian Journal of Statistics*, 1979, Vol. 6, pp. 65–70.
- [31] S. Marchand-Maillet, Y. M. Sharaiha, "A Minimum Spanning Tree Approach to Line Image Analysis," in Proceedings of 13th International Conference on Pattern Recognition (ICPR'96), 1996, pp. 225.
- [32] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured Data Regeneration with Network Coding in Distributed Storage Systems," in Proceedings of International Conference on Image Processing, Charleston, USA, 2009, pp. 481–484.
- [33] J. C. Gower, and G. J. S. Ross, "Minimum Spanning Trees and Single Linkage Cluster Analysis," *Journal of the Royal Statistical Society (Applied Statistics)*, 1969, Vol. 18, No. 1, pp. 54–64.
- [34] Z. Barzily, Z. Volkovich, B. Akteke-Öztürk, G. W. Weber, "On a Minimal Spanning Tree Approach in the Cluster Validation Problem," *Informatica*, 2009, Vol. 20, No. 2, pp. 187–202.
- [35] R. E. Osteen, and P. P. Lin, "Picture Skeletons Based on Eccentricities of Points of Minimum Spanning Trees," *SIAM Journal on Computing*, 1974, Vol. 3, pp. 23–40.
- [36] T. C. Chiang, C. H. Liu, and Y.M. Huang, "A Near-Optimal Multicast Scheme for Mobile Ad hoc Networks Using a Hybrid Genetic Algorithm," *Expert Systems with Applications*, 2007, Vol. 33 pp. 734–742.
- [37] G. Rodolakis, A. Laouiti, P. Jacquet, and A. M. Naimi, "Multicast Overlay Spanning Trees in Ad hoc Networks: Capacity bounds protocol design and performance evaluation," *Computer Communications*, 2008, Vol. 31, pp. 1400–1412.
- [38] H. Katagiri, E. B. Mermri, M. Sakawa and K. Kato, "A Study on Fuzzy Random Minimum Spanning Tree Problems through Possibilistic Programming and the Expectation Optimization Model," in Proceedings of the 47th IEEE International Midwest Symposium on Circuits and Systems, 2004.
- [39] H. Fangguo, and Q. Huan, "A Model and Algorithm for Minimum Spanning Tree Problems in Uncertain Networks," in Proceedings of the 3rd International Conference on Innovative Computing Information and Control (ICICIC'08), 2008.
- [40] T. A. Almeida, A. Yamakami, and M. T. Takahashi, "An Evolutionary Approach to Solve Minimum Spanning Tree Problem with Fuzzy Parameters," in Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, 2005.
- [41] K. R. Hutson, and D. R. Shier, "Minimum Spanning Trees in Networks with varying Edge Weights," *Annals of Operations Research*, Vol. 146, pp. 3–18, 2006.
- [42] K. R. Hutson, and D. R. Shier, "Bounding Distributions for the Weight of a Minimum Spanning Tree in Stochastic Networks," *Operations Research*, 2005, Vol. 53, No. 5, pp. 879–886.

- [43] J. Akbari Torkestani, and M. R. Meybodi, "A Learning Automata-based Cognitive Radio for Clustered Wireless Ad-Hoc Networks", *Journal of Network and Systems Management*, 2011, Vol. 19, No. 2, pp. 278-297.
- [44] J. Akbari Torkestani, and M. R. Meybodi, "A Learning automata-based Heuristic Algorithm for Solving the Minimum Spanning Tree Problem in Stochastic Graphs," *Journal of Supercomputing*, Springer Publishing Company, 2010, in press.
- [45] J. Akbari Torkestani, and M. R. Meybodi, "Learning Automata-Based Algorithms for Finding Minimum Weakly Connected Dominating Set in Stochastic Graphs," *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 2010, Vol. 18, No. 6, pp. 721-758.
- [46] J. Akbari Torkestani, and M. R. Meybodi, "Mobility-based Multicast Routing Algorithm in Wireless Mobile Ad Hoc Networks: A Learning Automata Approach," *Journal of Computer Communications*, 2010, Vol. 33, pp. 721-735.
- [47] J. Akbari Torkestani, and M. R. Meybodi, "A New Vertex Coloring Algorithm Based on Variable Action-Set Learning Automata," *Journal of Computing and Informatics*, 2010, Vol. 29, No. 3, pp. 1001-1020.
- [48] J. Akbari Torkestani, and M. R. Meybodi, "Weighted Steiner Connected Dominating Set and its Application to Multicast Routing in Wireless MANETs," *Wireless Personal Communications*, Springer Publishing Company, 2010, in press.
- [49] J. Akbari Torkestani, and M. R. Meybodi, "An Efficient Cluster-based CDMA/TDMA Scheme for Wireless Mobile AD-Hoc Networks: A Learning Automata Approach," *Journal of Network and Computer applications*, 2010, Vol. 33, pp. 477-490.
- [50] J. Akbari Torkestani, and M. R. Meybodi, "Clustering the Wireless Ad-Hoc Networks: A Distributed Learning Automata Approach," *Journal of Parallel and Distributed Computing*, 2010, Vol. 70, pp. 394-405.
- [51] J. Akbari Torkestani, and M. R. Meybodi, "An Intelligent Backbone Formation Algorithm in Wireless Ad Hoc Networks based on Distributed Learning Automata," *Journal of Computer Networks*, 2010, Vol. 54, pp. 826-843.
- [52] J. Akbari Torkestani, and M. R. Meybodi, "A Mobility-based Cluster Formation Algorithm for Wireless Mobile Ad Hoc Networks," *Journal of Cluster Computing*, Springer Publishing Company, 2011, in press.
- [53] J. Akbari Torkestani, and M. R. Meybodi, "A Cellular Learning Automata-based Algorithm for Solving the Vertex Coloring Problem," *Journal of Expert Systems with Applications*, 2011, in press.