J Supercomput (2012) 59:1035–1054
DOI 10.1007/s11227-010-0484-1

A learning automata-based heuristic algorithm for solving the minimum spanning tree problem in stochastic graphs

Javad Akbari Torkestani · Mohammad Reza Meybodi

Published online: 7 October 2010

Abstract During the last decades, a host of efficient algorithms have been developed for solving the minimum spanning tree problem in deterministic graphs, where the weight associated with the graph edges is assumed to be fixed. Though it is clear that the edge weight varies with time in realistic applications and such an assumption is wrong, finding the minimum spanning tree of a stochastic graph has not received the attention it merits. This is due to the fact that the minimum spanning tree problem becomes incredibly hard to solve when the edge weight is assumed to be a random variable. This becomes more difficult if we assume that the probability distribution function of the edge weight is unknown. In this paper, we propose a learning automata-based heuristic algorithm to solve the minimum spanning tree problem in stochastic graphs wherein the probability distribution function of the edge weight is unknown. The proposed algorithm taking advantage of learning automata determines the edges that must be sampled at each stage. As the presented algorithm proceeds, the sampling process is concentrated on the edges that constitute the spanning tree with the minimum expected weight. The proposed learning automata-based sampling method decreases the number of samples that need to be taken from the graph by reducing the rate of unnecessary samples. Experimental results show the superiority of the proposed algorithm over the well-known existing methods both in terms of the number of samples and the running time of algorithm.

J. Akbari Torkestani (✉)
Department of Computer Engineering, Islamic Azad University, Arak Branch, Arak, Iran
e-mail: j-akbari@iau-arak.ac.ir

M.R. Meybodi
Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, Iran
e-mail: mmeybodi@aut.ac.ir

M.R. Meybodi
Institute for Studies in Theoretical Physics and Mathematics (IPM), School of Computer Science, Tehran, Iran

Keywords Learning automata · Minimum spanning tree · Stochastic graph

1 Introduction

Spanning tree of a connected, undirected graph is a tree-based subgraph by which all the graph vertices are connected. A minimum spanning tree (MST) of an edge-weighted graph is a spanning tree having the minimum sum of edge weights among all the spanning trees. The weight assigned to each edge of the graph represents its cost, traversal time, or length depending on the context. The minimum spanning tree is an appealing structure in the design of the communication systems that economically connect spatially dispersed elements, computer networks, and other network-related problems. Minimum spanning trees also arise in more subtle applications in statistical cluster analysis [45, 46], data storage [47–49], picture processing [50], and speech recognition [51]. Due to the tremendous growth of the communication networks, the network applications of the minimum spanning trees have attracted a lot of attention during the last decades. For instance, the broadcasting problem, in which the same data must be sent to all the nodes within the network (one to all), can be simply modeled by the minimum spanning tree problem. The minimum spanning tree is also the optimal routing tree for data aggregation (all to one) in distributed environments. Besides, in some of the multicast routing protocols [1, 2], the minimum spanning tree is still one of the most effective and reliable methods to multicast the messages from a source node to a group of destinations. In most scenarios, the edge weight is assumed to be fixed, but such an assumption does not hold true in real world applications and the weights vary with time indeed. 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 MST problem in which the edge weights are not constant but random variables. There have been many studies of the minimum spanning tree problem dealing with the deterministic graphs and several renowned sequential algorithms have been designed such as Boruvka [3], Kruskal [4], and Prim [5] in which the MST problem can be solved in polynomial time. However, when the edge weight is allowed to be a random variable (or vary with time), the problem of finding the minimum spanning tree of the (stochastic) graph becomes incredibly difficult. This becomes more intractable, if the probability distribution function of the edge weight is assumed to be unknown.

Ishii et al. [6] 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 is presented to solve the latter problem. In this method, the probability distribution of the edge weight is assumed to be known. Ishii and Nishida [7] 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 [8] proposed a method for a stochastic spanning tree problem called interval elimination. In the proposed method, like Ishii et al. [6], the problem is first transformed into a deterministic equivalent problem and then solved. Mohd also introduced several modifications to the algorithm of Ishii et al. [6]. He showed that the modified algorithm is able to obtain

much better results in less time. Ishii and Matsutomi [9] presented a polynomial time algorithm to solve the problem stated in [6]. In this approach, the parameters of underling probability distribution of edge costs are assumed to be unknown, and so they are estimated by a confidence region from statistical data. In the proposed method, the problem is first transformed into a deterministic equivalent problem with a min-max type objective function and a confidence region of means and variances, since they assume that the random edge costs have normal distributions. In [52], Jain and Mamer proposed a method for estimation of the distribution of the MST weight in a stochastic network. They relaxed the condition that the random variable associated with the edge weight must be identically distributed. They obtained bounds on the distribution and the mean of the MST weight which is proved to be better than the naive bound obtained by solving the deterministic MST with expected edge weights. Alexopoulos and Jacobson's algorithm [10], which is hereafter referred to as ALJA, extended the partitioning technique considered in [40] to compute and bound specific values of the minimum spanning tree distribution in networks with independent, but not necessarily identically distributed, discrete edge weight random variables. Alexopoulos and Jacobson also proposed several methods to determine the probability that a given edge belongs to a minimum spanning tree. They demonstrated that the exact calculation of values of the minimum spanning tree distribution is NP-hard.

Katagiri et al. [11] examined the case where the edge weights are fuzzy random variables. They introduced a fuzzy-based approach to model the minimum spanning tree problem in case of fuzzy random weights. Almeida et al. [12] studied the minimum spanning tree problem with fuzzy parameters and proposed an exact algorithm to solve this problem. In [13], Hutson and Shier studied several approaches to find (or to optimize) the minimum spanning tree when the edges undergo the weight changes. Repeated Prim (RP) method, cut-set (CM) method, cycle tracing (CTM) method, and multiple edge (ME) sensitivity method are the proposed approaches to find the MST of the networks in which each edge weight can assume a finite number of distinct values. To approximate the expected weight of the optimal spanning tree, Hutson and Shier used the algebraic structure to describe the relationship between different edge-weight realizations of the network. They compared different approaches and showed that the multiple edge sensitivity method (ME), hereafter referred to as HUSH, outperforms the others in terms of the time complexity and the size of the constructed state space. Fangguo and Huan [14] considered the problem of minimum spanning trees in uncertain networks in which the edge weights are random variables. In [14], the concept of the expected minimum spanning tree is initially defined and a model of the problem is accordingly formulated. Based on this model, a hybrid intelligent algorithm as a combination of the genetic algorithm and stochastic simulation is proposed. In order to code the corresponding spanning tree for the genetic representation, the Prüfer encoding scheme that is able to represent all possible trees is employed. Dhamdhere et al. [15] and Swamy and Shmoys [16] formulated the stochastic minimum spanning tree problem as a stochastic optimization problem and proposed some approximation approaches to solve two and multistage stochastic optimization problems.

The major problem with the above mentioned stochastic minimum spanning tree algorithms is that they are practical when the probability distribution function (PDF)

of the edge weight is assumed to be known. While such an assumption does not hold true in realistic applications. In this paper, we propose a learning automata-based approximation algorithm for solving the minimum spanning tree problem in the stochastic graph, where the probability distribution function of the weight associated with the graph edge is unknown. In the proposed heuristic algorithm, by a learning automata-based sampling method, it is probabilistically decided whether an edge must be sampled or not. That is, each learning automaton which is assigned to a given graph node decides which incident edge must be sampled at each stage. In the course of the learning process, automata learn how to sample the edges along the minimum spanning tree with a higher probability. Hence, as the proposed algorithm approaches to the end, sampling process is concentrated on the edges by which the minimum spanning tree is constructed. In other words, the sampling process finally focuses on the spanning tree with the minimum expected weight. Such a probabilistic sampling method reduces the rate of unnecessary samples. To evaluate the performance of the proposed stochastic MST algorithm, the obtained results are compared with those of Alexopoulos and Jacobson [10] and Hutson and Shier [13], both in terms of the number of samples and running time of algorithm. The simulation experiments show that the proposed stochastic MST algorithm outperforms the algorithms proposed by Alexopoulos and Jacobson [10] and Hutson and Shier [13] in terms of all metrics of interest.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the minimum spanning tree problems, and describes the stochastic minimum spanning tree problem. Section 3 introduces the learning automata in a nut shell. In Sect. 4, the proposed learning automata-based algorithm is presented. Section 5 shows the performance of the proposed algorithm through simulation experiments and comparison with the best existing methods. Section 6 concludes the paper.

2 Minimum spanning tree problem

The Minimum Spanning Tree problem is a classical combinatorial optimization problem in graph theory. This problem is defined as to find the minimum weight spanning tree in a weighted graph. Many engineering problems such as the design of communication network, electric power system, and so on can be described by MST problem. The weight assigned to each edge of the network could represent its cost, traversal time, or length depending on the context. The minimum spanning tree problem can be generally subdivided into deterministic MST Problem and stochastic MST problem depending upon the edge weight is assumed to be fixed or a random variable. Figure 1 shows the new classification of the minimum spanning tree algorithms which is proposed in this paper.

A minimum spanning tree of a weighted, undirected graph G is a spanning tree of G whose edges sum to minimum weight. In other words, a minimum spanning tree is a tree formed from a subset of the edges in a given undirected graph, with two properties: first, it spans the graph, i.e., it includes every vertex in the graph, and then it is a minimum, i.e., the total weight of all the edges is as low as possible.

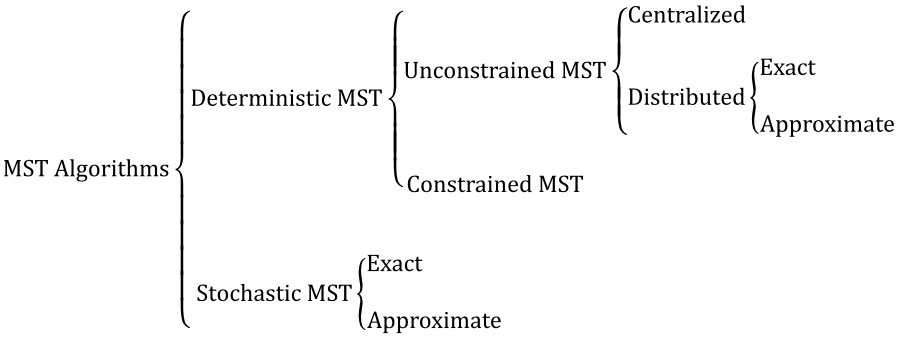

Fig. 1 A new classification of the MST algorithms

Definition 1 Let $G\langle V, E\rangle$ denotes an undirected graph consisting of vertex-set $V = \{v_1, v_2, \ldots, v_n\}$ and edge-set $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$. Subgraph $G'\langle V', E'\rangle$ of graph $G\langle V, E\rangle$ is a spanning tree, if we have

1) Subgraph G' is connected.
2) G' has the same vertex-set as G, i.e., $V' = V$.
3) $|E'| = n - 1$, where $|E'|$ denotes the cardinality of edge-set E'.

Definition 2 Let $G\langle V, E, W\rangle$ denotes an edge-weighted, undirected graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex-set, $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ is edge-set, and $W = \{w_1, w_2, \ldots, w_m\}$ is the set of weights associated with the edges. Let $T = \{\tau_1, \tau_2, \tau_3, \ldots\}$ denotes the set of possible spanning trees of graph G. Let w_j and $w_{\tau_i} = \sum_{\forall e_j \in \tau_i} w_j$ denote the weight associated with edge $e_j \in E$ and spanning tree $\tau_i \in T$. Therefore, spanning tree $\tau^* \in T$ is the minimum spanning tree (MST) of graph G, if $w_{\tau^*} = \min_{\forall \tau_i \in T}\{w_{\tau_i}\}$.

2.1 Deterministic minimum spanning tree problem

Deterministic MST problem deals with finding the minimum spanning tree of the graph where the weight associated with the graph edge is constant. Most researches study the minimum spanning tree problem when the edge weight is constant, and so a host of deterministic algorithms are available. As shown in Fig. 1, deterministic minimum spanning tree problem can be further subdivided as unconstrained and constrained MST problems. Constrained MST problem is a generalization of the MST problem in which some additional constraints must be satisfied.

Unconstrained MST algorithms are divided as centralized and distributed algorithms. Boruvka's algorithm [3], Kruskal's algorithm [4], and Prim's algorithm [5] are three well-known centralized MST algorithms. Distributed MST algorithms are further classified as exact and approximation algorithms. Gallager et al. [17], Spira [18], Dalal [19], Gafni [20], Awerbuch [21], Garay et al. [22], Elkin [23], and Kutten and Peleg [24] are representative exact solutions proposed for unconstrained MST problem. While the previous distributed algorithms deal with computing the exact MST, the next important question addressed in the literature concerns the study of distributed approximation of MST, i.e., constructing a spanning tree whose total weight is

near optimal. Peleg and Rabinovich [25], Elkin [26–28], and Maleq and Pandurangan [28] proposed several approximation algorithms for finding a near optimal solution to the MST problem.

Constrained minimum spanning tree problem is a bicriteria (or multicriteria) problem in which two (or more) parameters must be optimized. In other words, the constrained minimum spanning tree problem can be defined as a generalization of the MST problem in which some additional constraints are satisfied at the same time. Constrained minimum spanning tree problem is defined as follows.

Definition 3 Given a weighted, undirected graph $G\langle V, E, W, C\rangle$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex-set, $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ is edge-set, $W = \{w_1, w_2, \ldots, w_m\}$ is the set of weights associated with the edges, and $C = \{c_1, c_2, \ldots, c_m\}$ denotes the (additional) constraints imposed to the edges. The constrained minimum spanning tree problem (CMSTP) can be formulated as the following optimization problem.

$$\min \sum_{\forall e_j \in T} w_j \tag{1}$$

subject to

$$\sum_{\forall e_j \in T} c_j \leq L, \tag{2}$$

where L is the imposed constraint and T is the solution to the CMSTP. That is, the constrained minimum spanning tree problem is to find the spanning tree with the minimum total weight and the total constraint at most L.

Aggarwal et al. [29] proved that the constrained minimum spanning tree problem is a weakly NP-hard problem. Representative constrained minimum spanning tree problems studied in the literature include Bounded Diameter Minimum Spanning Tree (BDMST) [30], Degree Constrained Minimum Spanning Tree (DCMST) [31], Capacitated Minimum Spanning Tree (CMST) [32], Generalized Minimum Spanning Tree (GMST) [33], Delay-Constrained Minimum Spanning Tree [34, 35], and Hop-Constrained Minimum Spanning Tree (HMST) [36]. Since the constrained minimum spanning tree problem is an NP-hard problem, heuristic methods, such as tabu search [37], ant colony optimization [31], genetic algorithms [38, 39], and fuzzy-based algorithms [32], have been extensively used by the researchers for solving this complex optimization problem.

2.2 Stochastic minimum spanning tree problem

As mentioned above, a deterministic MST algorithm aims at finding the minimum spanning tree of the graph, where the edge weight is assumed to be fixed, while a stochastic minimum spanning tree algorithm deals with the graph edge whose weight is a random variable. In most scenarios, it is assumed that the edge weights are fixed, but this is not always true. For example, links in a communication network can malfunction or degrade as a result of congestion, accidents, weather, etc. More generally,

the edges of a time-varying network can assume several states. Therefore, a deterministic graph is not able to realistically model the characteristics of such networks, and so the network topology should be modeled by a stochastic graph. As mentioned before, several algorithms have been proposed to solve the minimum spanning tree problem, where the network parameters are deterministic. However, finding the minimum spanning tree becomes considerably harder when the graph is stochastic. In what follows, we define the stochastic minimum spanning tree problem and review the stochastic MST algorithms in a nut shell.

Definition 4 A stochastic edge-weighted graph G is defined by a triple $\langle V, E, W\rangle$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the vertex-set, $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ denotes the edge-set, and $W = \{w_1, w_2, \ldots, w_m\}$ denotes the set of weights associated with the edge-set such that positive random variable w_i is the weight of edge $e_i \in E$.

Definition 5 Let $G\langle V, E, W\rangle$ denotes a stochastic edge-weighted graph, and $T = \{\tau_1, \tau_2, \tau_3, \ldots\}$ denotes the set of possible spanning trees of the stochastic graph $G\langle V, E, W\rangle$. Let $\bar{w}_{\tau_i}$ denotes the expected weight of spanning tree τ_i. The stochastic MST (SMST) is defined as a stochastic spanning tree with the minimum expected weight. That is, stochastic spanning tree $\tau^* \in T$ is the stochastic minimum spanning tree if and only if $\bar{w}_{\tau^*} = \min_{\forall \tau_i \in T}\{\bar{w}_{\tau_i}\}$.

Several authors have examined network optimization problems where the edge weights are determined by independent (though not necessarily identically distributed) discrete random variables. However, in stochastic graphs, the MST problem has not received the attention it deserves. The existing stochastic minimum spanning tree algorithms are further subdivided as exact [6–10] and approximation [11, 12, 14–16] algorithms. Due to the hardness of the stochastic minimum spanning tree problem for general stochastic graphs, exact algorithms are only feasible for small graphs, while very large graphs often arise in realistic applications. Therefore, polynomial time approximation algorithms have been also proposed for finding a near optimal solution of the stochastic minimum spanning tree problem.

The following briefly describes two methods proposed for solving the minimum spanning tree problem in stochastic graphs with which our proposed algorithm is compared. In [40], an efficient heuristic approach was presented by Doulliez and Jamoulle for computing the probabilistic measures of the multistage systems. The proposed method is based on the iteratively partitioning the state space of the system. The proposed decomposition technique was first proposed for solving the stochastic maximum flow problem in networks with discrete arc capacities where all the flow requirements must be satisfied. Alexopoulos and Jacobson [10] enhanced the efficiency of the state space partitioning technique presented in [40] by extending and enriching its theoretical foundations. They applied the extended partitioning technique for computing the probability distribution of the weight of the minimum spanning trees in stochastic graphs with independent, but not necessarily identically distributed, discrete edge weight random variables. They proved that the exact calculation of the parameters of the probability distribution of the weight of the minimum spanning tree is known to be NP-hard. The number of iterations of the proposed algorithm is

typically small for moderate size problems, however, for large graphs Alexopoulos and Jacobson's algorithm may cause an intractable computational cost. Therefore, the running time of their proposed algorithm is not necessarily polynomial. The state space partitioning technique upon which the Alexopoulos and Jacobson's algorithm is based computes the probabilistic measure (the probability distribution of the weight of the stochastic MST) by dividing all the possible edge-weight realizations (the entire state space of the problem) iteratively into subsets with known contribution to the probabilistic measure and subsets with unknown contribution. This continues until no sets with unknown contribution remain to be processed. In fact, the partitioning technique is very similar to a factoring or branch-and-bound procedure, where the nodes of the search tree are the sets with unknown contribution. At any iteration, the bounds on the probabilistic measure can be computed. As the algorithm proceeds, the bounds get more tightened and finally equal to the value of the probabilistic measure.

Hutson and Shier [13] considered several approaches to solve the minimum spanning tree problem in networks in which the edge weight can assume a finite number of distinct values. In [13], the authors believe that even for small graphs the state space of the problem (i.e., the number of possible edge-weight realizations) is massive. Therefore, to alleviate the negative impacts of the huge state space, they propose a systematic way based on Hasse diagram for generating the state space which avoids generating unnecessary and repetitive states. In this method, the algebraic structure of the underlying Hasse diagram is exploited to find the relationship between the different edge-weight realizations of the stochastic network. This algebraic structure represents a graphical form of the state space in which a state is represented by a node and an edge represents a change to the next largest weight of exactly one edge. Then a rooted spanning tree and a traversal algorithm of that tree combine to generate each state without repetition. Hutson and Shier then proposed several approaches for calculating the minimum spanning trees associated with the above constructed state space. The proposed approaches are repeated prim (RP) method, cut-set (CT) method, cycle tracing method (CTM), and multiple edge (ME) sensitivity method. They theoretically analyzed the complexities of different approaches and showed that RP and CM are always faster than CTM. Such a result is expected since RP and CM have worst case running times of order $O(n^2N)$ and $O(mN)$, respectively, which dominate the complexity of order $O(nmN)$ for CTM, where n is the number of nodes, m is the number of edges, and N denotes the number of states. They also show that ME has a worst case running time of $O(mN)$. Hutson and Shier also conducted several experiments to show the performance of the proposed algorithms. The obtained results show that the multiple edge sensitivity method (ME) outperforms the others in terms of the time complexity and the size of the constructed state space. Therefore, to show the superiority of our method over the proposed approaches, we compare it with ME. The main problem with the proposed approaches in [13], specifically ME method, is that to reduce the complexities of the method they take into consideration only a small subset of all the possible realizations of the stochastic network. Such a reduced state space may cause the missing of the optimal solution or even near optimal solutions. Hence, these approaches do not assure the minimum expected solution.

3 Learning automata

A learning automaton [43, 44] 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. 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.

Learning automata have been found to be useful in systems where incomplete information about the environment exists [60]. Learning automata are also proved to perform well in complex, dynamic and random environments with a large amount of uncertainties. A group of learning automata can cooperate to cope with many hard-to-solve problems. To name just a few, learning automata have a wide variety of applications in combinatorial optimization problems [53, 55], computer networks [54, 56–59, 66], queuing theory [61], signal processing [62], information retrieval [63], adaptive control [64], and pattern recognition [65].

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, \ldots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, \ldots, c_r\}$ 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 nonstationary environment. The environments depending on the nature of the reinforcement signal $\underline{\beta}$ 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 [0, 1].

Learning automata can be classified into two main families [43]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \underline{\beta}, \underline{\alpha}, T \rangle$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ 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 $\alpha_i(k) \in \underline{\alpha}$ and $\underline{p}(k)$ 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 $\underline{p}(k)$ is updated by the linear learning algorithm given in (3), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in (4) if the taken action is penalized.

$$p_j(k+1) = \begin{cases} p_j(k) + a[1 - p_j(k)], & j = i \\ (1-a)p_j(k), & \forall j \neq i \end{cases} \tag{3}$$

$$p_j(k+1) = \begin{cases} (1-b)p_j(k), & j = i \\ \left(\frac{b}{r-1}\right) + (1-b)p_j(k), & \forall j \neq i \end{cases} \tag{4}$$

If $a = b$, the recurrence equations (3) and (4) 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.

3.1 Variable action set learning automata

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 [42] 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, $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. $A = \{A_1, A_2, \ldots, A_m\}$ 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 $\Psi(k) = \{\Psi_1(k), \Psi_2(k), \ldots, \Psi_m(k)\}$ defined over the possible subsets of the actions, where $\Psi_i(k) = \text{Prob}[A(k) = A_i \mid A_i \in A, 1 \leq i \leq 2^n - 1]$. $\hat{p}_i(k) = \text{Prod}[\alpha(k) = \alpha_i \mid A(k), \alpha_i \in A(k)]$ is the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and also $\alpha_i \in A(k)$. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = p_i(k)/K(k) \tag{5}$$

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 $p_i(k) = \text{Prod}[\alpha(k) = \alpha_i]$.

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 k. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in (5). 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 probability vector of the actions 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 [42].

4 The proposed stochastic MST algorithm

As mentioned earlier, many studies have been conducted on deterministic minimum spanning tree problem, but the stochastic minimum spanning tree problem has not received the attention it deserves. On the other side, due to the stochastic nature of

the real world network applications, deterministic algorithms are not capable of finding the minimum spanning tree in such stochastic networks. Therefore, in this paper, we propose a learning automata-based approximation algorithm called LASMSTA (short for learning automata-based stochastic minimum spanning tree algorithm) for finding the optimal solution of the stochastic minimum spanning tree problem, where the probability distribution function of the edge weight is unknown. The deterministic case of the optimum spanning tree problem has been well studied, and until now several powerful polynomial time algorithms have been proposed. But when the edge weight varies with time, the optimum solution of the MST problem is extremely hard to find. The aim of this paper is to show the capabilities of the learning automata for solving such a difficult problem. The proposed algorithm is based on a sampling method in which at each stage a set of learning automata determines which edges must be sampled. This sampling method may result in decreasing unnecessary samples, and hence decreasing the running time of algorithm.

Let $G\langle V, E, W\rangle$ denotes the input stochastic graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex-set, $E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$ is the edge-set, and matrix W denotes the weights associated with the edge-set. In this algorithm, a network of learning automata isomorphic to the stochastic graph is initially formed by equipping each node of the graph with a L_{R_I} learning automaton. The resulting network can be described by a triple $\langle \underline{A}, \underline{\alpha}, \underline{W}\rangle$, where $\underline{A} = \{A_1, A_2, \ldots, A_n\}$ denotes the set of the learning automata, $\underline{\alpha} = \{\underline{\alpha}_1, \underline{\alpha}_2, \ldots, \underline{\alpha}_n\}$ denotes the set of all possible actions in which $\underline{\alpha}_i = \{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^{r_i}\}$ defines the set of actions that can be chosen by learning automata A_i (for each $\underline{\alpha}_i \in \underline{\alpha}$) and r_i is the cardinality of action-set $\underline{\alpha}_i$. Edge $e_{(i,j)}$ corresponds either to the action α_i^j of the learning automata A_i or to the action α_j^i of the learning automata A_j. That is, each learning automaton can select each of its incident edges as an action. Choosing action α_i^j by automaton A_i adds edge $e_{(i,j)}$ to the minimum spanning tree. Weight $w_{i,j}$ is the weight associated with edge $e_{(i,j)}$ and assumed to be a positive random variable with an unknown probability distribution.

In the proposed algorithm, each learning automaton can be in one of two modes active and passive. All learning automata are initially set to the passive state. The proposed algorithm consists of a number of stages and at each stage one of the possible spanning trees is randomly constructed. The proposed algorithm is based on the distributed learning automata, and to explore the spanning trees, it traverses the distributed learning automata by the backtracking technique. Each stage of the LASMSTA algorithm is initiated by choosing one of the graph vertices at random. The learning automaton corresponding to the selected vertex is activated and chooses one of its actions based on its action probability vector. The edge corresponding to the selected action is added to the spanning tree which is currently being formed. The weight associated with the selected edge is added to the total weight of spanning tree as well. To avoid the loops in the tree, each passive learning automaton prunes its action-set (or scales up its action probability vector) by disabling the actions corresponding to the edges selected so far or the edges by which a cycle may be formed. Then the learning automaton which is at the other end of the selected edge is activated. It chooses one of its actions as the previous activated automata did. The sequential activation process of learning automata (or selecting tree edges) is repeated until either a spanning tree

is constructed, or no more actions can be taken by the currently active learning automaton. In the former case, the current stage is successfully completed by finding a solution to the minimum weight spanning tree problem (this occurs when the number of selected edges is greater than or equal to $(n-1)$, where n denotes the cardinality of the vertex-set), and in the latter case, the proposed algorithm traces the path induced by the activated learning automata back for finding a learning automaton with available actions. The learning automaton which is found in the backtracking process is activated again. The action-set of such an automaton has to be updated by disabling its last selected action. Now, the reactivated automaton resumes the current stage by choosing one of its possible actions as described above. The learning automaton activation (or reactivation) process is continued until formation a spanning tree. The backtracking technique proposed in this paper assures that a spanning tree will be constructed at each stage of algorithm. By the backtracking technique, each learning automaton may activate more than one of its neighbors at each stage. That is, more than one action can be chosen by each learning automaton.

As mentioned earlier, the corresponding edge is added to the spanning tree, once an action is chosen by a learning automaton. The weight associated with the selected edge is also added to the total weight of the spanning tree. Since the weight associated with the graph edge is assumed to be a positive random variable, a particular spanning tree may experience a different weight at each stage. Therefore, the proposed algorithm deals with the average weight of the spanning trees rather than their weight at each stage. To do so, at the end of stage k, the average weight of the selected spanning tree is computed as follows:

We suppose that spanning tree τ_i is selected at stage k. The average weight of spanning tree τ_i until stage k is computed as

$$\bar{w}_{\tau_i}^{k} = \frac{1}{k_i} \sum_{j=1}^{k_i} w_{\tau_i}^{j} \tag{6}$$

where k_i denotes the number of times spanning tree τ_i is constructed until stage k, and $w_{\tau_i}^{j}$ denotes the weight of the jth sample of spanning tree τ_i, which is defined as

$$w_{\tau_i}^{j} = \sum_{\forall (s,t) \in \tau_i} w_{e_{(s,t)}}^{j} \tag{7}$$

where $w_{e_{(s,t)}}^{j}$ denotes the weight of edge $e_{(s,t)}$ as a part of the jth sample taken from spanning tree τ_i.

To guarantee the convergence of the proposed algorithm to the optimal solution (i.e., minimum spanning tree), the average weight of the constructed spanning tree has to be compared with the dynamic threshold, T_k, at each stage. At stage $k > 1$, the dynamic threshold is calculated as

$$T_k = \frac{1}{r} \sum_{i=1}^{r} \bar{w}_{\tau_i}^{k} \tag{8}$$

where r denotes the number of all spanning trees explored until stage k.

Algorithm LASMSTA The proposed stochastic Minimum Spanning Tree algorithm

Input: Graph $G\langle V, E, W\rangle$, Stop Threshold S
Output: The minimum spanning tree
Assumptions
Let τ denotes the selected tree
Begin Algorithm
$k \leftarrow 0$, $T_k \leftarrow 0$
Repeat
  $\tau \leftarrow \emptyset$, $w_\tau \leftarrow 0$
  The first automaton is randomly selected, denoted as A_j and activated
  Repeat
    If A_i has no possible actions **Then**
      Path induced by activated automata is traced back to find automaton with available actions
      The found learning automaton is denoted as A_i
    End If
    Automaton A_i chooses one of its actions (say action α_i^j)
    $\tau \leftarrow \tau + \{e_{(A_i,A_j)}\}$, $w_\tau \leftarrow w_\tau + \{w_{e_{(A_i,A_j)}}\}$
    Each automaton prunes its action-set to avoid the loop
    Automaton A_j is activated
    Set A_i to A_j
  Until $|\tau| \geq |V| - 1$
  Compute the average weight of the selected spanning tree and denote it $\bar{w}_\tau$
  If $\bar{w}_\tau < T_{k-1}$ **Then**
    Reward the selected actions of the activated automata along the spanning tree
  Else
    Penalize the selected actions of the activated automata along the spanning tree
  End If
  $T_k \leftarrow [(k-1)T_{k-1} + \bar{w}_\tau]/k$
  $k \leftarrow k + 1$
  Enable all the disabled actions
Until the probability of finding a MST is greater than S
End Algorithm

Fig. 2 Pseudo code of the proposed stochastic MST algorithm

At each stage, the average weight of the selected spanning tree is compared with the dynamic threshold. All activated learning automata reward their chosen actions, if the average weight of the selected spanning tree is less than or equal to the dynamic threshold. They penalize the taken actions otherwise. Since each learning automaton updates its action probability vector by using a L_{R-I} learning algorithm, the probability vectors remain unchanged when the learning automata are penalized. At the end of each stage, the disabled actions must be enabled again and the action probabilities are rescaled as described on variable action learning automata in Sect. 3.1. The process of constructing the spanning trees and updating the action probabilities is repeated until the choice probability of the constructed spanning tree is greater than a certain threshold S which is called stop threshold. The choice probability of a spanning tree is defined as the product of the probability of choosing the selected edges.

The spanning tree which is selected just before the algorithm stops is the spanning tree with the minimum expected weight among all the spanning trees of the stochastic graph. Figure 2 shows the pseudocode of the proposed Stochastic MST algorithm.

5 Experimental results

To study the performance of the proposed stochastic minimum spanning tree algorithm, we have conducted several simulation experiments on four well-known stochastic benchmark graphs borrowed from [10, 13]. The running time of algorithm and the number of samples taken from the stochastic graph (i.e., sampling rate) are our metrics of interest. To show the outperformance of our proposed algorithm, the obtained results are compared with those of algorithms proposed by Alexopoulos and Jacobson [10] and Hutson and Shier [13]. All algorithms are tested on two sparse graphs called Alex1 and Alex2 as well as two complete graphs with 5 and 6 vertices called K_5 and K_6, respectively. Alex1 comprises 8 nodes and 14 edges, and Alex 2 has 9 nodes and 15 edges. The discrete random variables associated with the edge weight of Alex1 and Alex2 have two and three states in mode A and B, respectively. The probability distributions of the random weights assigned to the edges of Alex1 and Alex2 given in [41] tend toward the smaller edge weights. That is, 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 other two benchmark stochastic graphs on which we tested the studied algorithms are two complete graphs K_5 and K_6 given in [13]. All stochastic MST algorithms were tested on benchmarks graphs K_5 and K_6 in two different modes and the obtained results reported in Tables 1 and 2. In the former mode denoted as E1, the distribution of the edge weight has a small variance, and in the latter mode specified as E2 the variance of the distribution associated with the edge weight is large. The random variables assigned to the edge weight of K_5 and K_6 have four and three states, respectively.

In learning automata-based algorithms, choosing the (proper) learning rate is the most challenging issue. From the learning automata theory, it is concluded that in our proposed algorithm the costs of the learning process increases and the expected weight of the spanning tree decreases (converges to the optimal one) as the learning rate decreases. That is, the solution optimality is inversely proportional to the learning rate. Such a conclusion can be also drawn from the results shown in Table 1. This property enables us to make a trade-off between the costs (running time and sampling rate) of the proposed algorithm and the optimality of the obtained solution by a proper choice of the learning rate. This means that the complexity of the proposed algorithm can be accommodated to the required optimality of the solution. To estimate the optimality of the solution, we calculate the percentage of the converged runs (PCR) to the expected weight of the minimum spanning tree for different values of learning rate. PCR is measured for 100 independent runs. To find an appropriate learning rate, we compute the running time (RT) of algorithm (in seconds), the total number of samples taken from the graph edges (i.e., the sampling rate of algorithm) (SR), and the percentage of the converged runs (PCR) as the learning rate of algorithm varies from

Table 1 The performance evaluation of the proposed algorithm for different learning rates

Learning rate	RT	SR	PCR
0.05	5.9029	5890	100
0.06	3.7890	4921	100
0.07	2.3400	3764	100
0.08	1.9097	3118	100
0.09	0.9412	2912	100
0.10	1.0350	2101	100
0.15	0.2100	1623	98
0.20	0.1050	1234	97
0.25	0.0970	1031	96
0.30	0.0783	989	93
0.35	0.0611	711	90
0.40	0.0510	590	88
0.45	0.0123	357	85
0.50	0.0098	210	81

Table 2 The average running time (RT) of different algorithms (in seconds)

Graph	Vertices	Edges	ALJA	HUSH	LASMSTA
Alex1-A	8	14	7.412	3.110	1.035
Alex2-A	9	15	15.70	7.231	1.482
Alex1-B	8	14	19.42	19.45	1.982
Alex2-B	9	15	34.78	28.12	2.609
K_5-E1	5	10	18.38	8.450	1.749
K_5-E2	5	10	30.29	12.98	2.198
K_6-E1	6	15	85.21	53.87	5.054
K_6-E2	6	15	101.2	69.44	8.290

0.05 to 0.50 for Alex1. The obtained results are shown in Table 1. Since the proposed algorithm aims at finding the minimum solution, from the obtained results, it can be observed that the proposed algorithm always converges to the minimal solution with the minimum number of iterations and minimum number of samples, if the learning rate is set to 0.1.

As discussed earlier, the simulation results given in Table 1 reveal that the running time of the proposed algorithm is inversely proportional to the learning rate. The results also show that the sampling rate and the convergence rate to the optimal solution increases as the learning rate decreases.

In learning automata-based algorithms, the convergence rate to the optimal solution is inversely proportional and the convergence speed is directly proportional to the learning rate. This is due to the fact that a learning automata-based algorithm with a small enough learning rate is capable of exploring almost all possible solutions, and so finds the best one. In fact, the costs of a learning automata-based algorithm (e.g., computational or communicational costs) increase as the learning rate decreases. As shown in Table 1, the running time of algorithm (which is the inverse of the conver-

Table 3 The average sampling rate (SR) of different algorithms

Graph	Vertices	Edges	ALJA	HUSH	LASMSTA
Alex1-A	8	14	24,117	10,760	2101
Alex2-A	9	15	35,982	23,412	2760
Alex1-B	8	14	76,980	44,902	3208
Alex2-B	9	15	103,492	80,981	3912
K_5-E1	5	10	142,981	53,902	3032
K_5-E2	5	10	192,091	62,411	4490
K_6-E1	6	15	210,376	60,787	6210
K_6-E2	6	15	311,442	71,192	6341

gence speed) increases as the learning rate becomes smaller. From Table 1, it can be seen that the sampling rate is directly proportional to the running time and increases as the learning rate increases. On the other hand, the optimality of the response (e.g., the percentage of the converged runs (PCR)) increases as the learning rate decreases. The results given in Table 1 also show that PCR increases as the learning rate decreases. This is because the number of stages of algorithm increases and so the algorithm has enough time to find the optimal (or very near to optimal) solution.

In all learning automata-based experiments conducted in this paper, the reinforcement scheme under which the action probability vector of the learning automata is updated is a linear reward-inaction (L_{R-I}) algorithm with learning rate 0.1. The stop threshold S is set to 0.95. This means that the proposed algorithm stops if the probability with which a spanning tree is selected becomes greater than or equal to 0.95. Each algorithm is tested on all the above mentioned variations of the stochastic benchmark graphs Alex1, Alex2, K_5, and K_6 and the results are summarized in Tables 2 and 3. The results reported in these tables are average over 100 runs. Table 2 shows the running time of each algorithm (in seconds), and Table 3 represents the total number of samples need to be taken from the graph edges (i.e., sampling rate of algorithm) by each algorithm.

From the numerical results reported in Table 2, it can be concluded that graphs K_5 and K_6 are more time consuming than Alex1 and Alex2. This is due to the fact that a larger number of samples must be taken from the large state space of the complete graphs (i.e., K_5 and K_6) for convergence to the optimal spanning tree even for smaller vertex-sets. This is also because of the probability distribution function of the random variable associated with the edge weight.

Table 2 shows the average amount of time consumed by each algorithm to solve the minimum spanning tree problem for every above mentioned stochastic benchmark graphs. Comparing the running time of ALJA, HUSH, and LASMSTA given in Table 2, it is observed that the time complexity of our proposed algorithm (LASMSTA) is significantly shorter than that of HUSH and ALJA. This is because of the fact that, unlike ALJA and HUSH, the proposed algorithm is not based on the construction of the extremely large state space of the stochastic problem. Furthermore, LASMSTA removes the non-optimal edges (i.e., the edges that are not along the branches of the optimal spanning tree) from the sampling process for the next stages. That is, as LASMSTA proceeds the edges or branches which do not belong to the

optimal spanning tree are pruned. As mentioned earlier, the process of constructing the problem state space is extremely time consuming for moderate size or even small stochastic graphs, and becomes an intractable problem in large networks. Therefore, ALJA and HUSH take too much time for the state space construction phase. In addition, finding the optimal tree form the huge state space requires a long time. Hence, it is expected that the running time of HUSH and ALJA will be much longer than that of LASMSTA. The obtained results given in Table 1 confirm this, showing that HUSH lags far behind LASMSTA, and ALJA takes a longer time as compared with HUSH. This is because HUSH considers only a small subset of all the possible realizations of the stochastic problem state space, and avoids generating unnecessary and repetitive states. The reduced state space shortens the running time of HUSH in comparison with ALJA. On the other hand, reducing the number of states may cause the missing of the optimal state in HUSH. As a result, the probability of finding the optimal solution in HUSH is smaller than that of ALJA. Form Table 1, it can be observed that LASMSTA always converges to the optimal solution (i.e., spanning tree with the minimum expected weight) for learning rates less than 0.15, while the rate of the convergence to the optimal solution is at most 90% for HUSH and at most 94% for ALJA. Putting together the higher convergence rate and shorter running time of the proposed algorithm reveals its outperformance better.

We also conducted several simulation experiments to measure the average sampling rate (i.e., the average number of samples need to be taken from the stochastic graph) of ALJA, HUSH, and LASMSTA on different variations of the stochastic benchmark graphs Alex1, Alex2, K_5, and K_6. The obtained results are summarized in Table 3. Comparing the results given in this table, it is obvious that LASMSTA considerably outperforms the other algorithms in terms of the sampling rate. This is because the proposed learning automata-based algorithm removes the edges or branches of the graph which do not belong to the optimal spanning tree from the sampling process, and so is concentrated on the edges that construct the spanning tree with the minimum expected weight. Therefore, the proposed algorithm reduces the rate of unnecessary samples meaningfully. The results also show that the sampling rate of HUSH is very smaller in contrast with ALJA, specifically for the complete benchmark graphs. This is due to the fact that HUSH reduces the size of the problem state space by avoiding the unnecessary and repetitive states. However, sampling from the very huge state space of the stochastic problem which is constructed in ALJA and HUSH causes a much higher sampling rate compared to LASMSTA. In addition to the lower sampling rate, LASMSTA has a higher convergence rate (to the minimal solution) in comparison with HUSH and ALJA.

6 Conclusion

In this paper, we first proposed a classification of the minimum spanning tree problems, and then a learning automata-based heuristic algorithm for solving the minimum spanning tree problem in a stochastic graph where the probability distribution function of the edge weight is unknown. At each stage of the proposed algorithm, the edges that must be sampled are determined by the learning automata assigned

to the graph vertices. As the proposed algorithm proceeds, the sampling process is focused on the edges by which the spanning tree with the minimum expected weight is constructed. Therefore, the proposed algorithm significantly decreases the rate of unnecessary samples. To show the performance of the proposed algorithm, we conducted several simulation experiments and compared the obtained results with those of the best existing methods. The results show the superiority of our proposed algorithm over the others in terms of the convergence rate to the optimal solution, running time of algorithm, and the sampling rate.

References

1. Chiang TC, Liu CH, Huang YM (2007) A near-optimal multicast scheme for mobile ad hoc networks using a hybrid genetic algorithm. Expert Syst Appl 33:734–742
2. Rodolakis G, Laouiti A, Jacquet P, Naimi AM (2008) Multicast overlay spanning trees in ad hoc networks: capacity bounds protocol design and performance evaluation. Comput Commun 31:1400–1412
3. Boruvka O (1926) Ojistém problému minimálním (About a certain minimal problem). Praca Moravske Prirodovedecke Spolecnosti 3:37–58
4. Kruskal JB (1956) On the shortest spanning sub tree of a Graph and the traveling salesman problem. In: Proceedings of the American mathematical society 7(1):748–750
5. Prim RC (1957) Shortest connection networks and some generalizations. Bell Syst Tech J 36:1389–1401
6. Ishii H, Shiode S, Nishida T, Namasuya Y (1981) Stochastic spanning tree problem. Discrete Appl Math 3:263–273
7. Ishii H, Nishida T (1983) Stochastic bottleneck spanning tree problem. Networks 13:443–449
8. Mohd IB (1994) Interval elimination method for stochastic spanning tree problem. Appl Math Comput 66:325–341
9. Ishii H, Matsutomi T (1995) Confidence regional method of stochastic spanning tree problem. Math Comput Model 22(19–12):77–82
10. Alexopoulos C, Jacobson JA (2000) State space partition algorithms for stochastic systems with applications to minimum spanning trees. Networks 35(2):118–138
11. Katagiri H, Mermri EB, Sakawa M, Kato K (2004) 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
12. Almeida TA, Yamakami A, Takahashi MT (2005) 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
13. Hutson KR, Shier DR (2006) Minimum spanning trees in networks with varying edge weights. Ann Oper Res 146:3–18
14. Fangguo H, Huan Q (2008) 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)
15. Dhamdhere K, Ravi R, Singh M (2005) On two-stage stochastic minimum spanning trees. Springer, Berlin, pp 321–334
16. Swamy C, Shmoys DB (2006) Algorithms column: approximation algorithms for 2-stage stochastic optimization problems. ACM SIGACT News 37(1):1–16
17. Gallager RG, Humblet PA, Spira PM (1983) A distributed algorithm for minimum weight spanning trees. ACM Trans Program Lang Syst 5:66–77
18. Spira P (1977) Communication complexity of distributed minimum spanning tree algorithms. In: Proceedings of the second Berkeley conference on distributed data management and computer networks
19. Dalal Y (April 1977) Broadcast protocols in packet switched computer networks. Technical Report 128. Department of Electrical Engineering, Stanford University, Stanford
20. Gafni E (1985) Improvements in the time complexity of two message-optimal election algorithms. In: Proceedings of the 4th symposium on principles of distributed computing (PODC), pp 175–185

21. Awerbuch B (1987) 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), pp 230–240
22. Garay J, Kutten S, Peleg D (1998) A sublinear time distributed algorithm for minimum-weight spanning trees. SIAM J Comput 27:302–316
23. Kutten S, Peleg D (1998) Fast distributed construction of k-dominating sets and applications. J Algorithms 28:40–66
24. Elkin M (2004) A faster distributed protocol for constructing minimum spanning tree. In: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA), pp. 352–361
25. Peleg D, Rabinovich V (1999) A near-tight lower bound on the time complexity of distributed MST construction. In: Proceedings of the 40th IEEE symposium on foundations of computer science (FOCS), pp 253–261
26. Elkin M (2004) 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), pp 331–340
27. Elkin M (2004) An overview of distributed approximation. ACM SIGACT News 35(4):40–57
28. Khan M, Pandurangan G (2006) A fast distributed approximation algorithm for minimum spanning trees. In: Proceedings of the 20th international symposium on distributed computing (DISC)
29. Aggarwal V, Aneja Y, Nair K (1982) Minimal spanning tree subject to a side constraint. Comput Oper Res 9:287–296
30. Gruber M, Hemert J, Raidl GR (2006) Neighborhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA and ACO. In: Proceedings of genetic and evolutionary computational conference (GECCO'2006)
31. Bui TN, Zrncic CM (2006) An ant-based algorithm for finding degree-constrained minimum spanning tree. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp 11–18
32. Oncan T (2007) Design of capacitated minimum spanning tree with uncertain cost and demand parameters. Inf Sci 177:4354–4367
33. Oencan T, Cordeau JF, Laporte G (2008) A tabu search heuristic for the generalized minimum spanning tree problem. Eur J Oper Res 191(2):306–319
34. Parsa M, Zhu Q, Garcia-Luna-Aceves JJ (1998) An iterative algorithm for delay-constrained minimum-cost multicasting. IEEE/ACM Trans Netw 6(4):461–474
35. Salama HF, Reeves DS, Viniotis Y (1997) The Delay-constrained minimum spanning tree problem. In: Proceedings of the second IEEE symposium on computers and communications, pp 699–703
36. Gouveia L, Simonetti L, Uchoa E (2009) Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. J Math Program (in press)
37. Sharaiha YM, Gendreau M, Laporte G, Osman IH (1998) A tabu search algorithm for the capacitated shortest spanning tree problem. Networks 29(3):161–171
38. Hanr L, Wang Y (2006) A novel genetic algorithm for degree-constrained minimum spanning tree problem. Int J Comput Sci Netw Secur 6(7A):50–57
39. Krishnamoorthy M, Ernst A (2001) Comparison of algorithms for the degree constrained minimum spanning tree. J Heurist 7:587–611
40. Doulliez P, Jamoulle E (1972) Transportation networks with random arc capacities. RAIRO Oper Res 3:45–60
41. Hutson KR, Shier DR (2005) Bounding distributions for the weight of a minimum spanning tree in stochastic networks. Oper Res 53(5):879–886
42. Thathachar MAL, Harita BR (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern SMG17:1095–1100
43. Narendra KS, Thathachar KS (1989) Learning automata: an introduction. Printice-Hall, New York
44. Lakshmivarahan S, Thathachar MAL (1976) Bounds on the convergence probabilities of learning automata. IEEE Trans Syst Man Cybern SMC-6:756–763
45. Gower JC, Ross GJS (1969) Minimum spanning trees and single linkage cluster analysis. J R Stat Soc 18(1): 54–64
46. Barzily Z, Volkovich Z, Akteke-Öztürk B, Weber GW (2009) On a minimal spanning tree approach in the cluster validation problem. Informatica 20(2):187–202
47. Marchand-Maillet S, Sharaiha YM (1996) A minimum spanning tree approach to line image analysis. In: Proceedings of 13th international conference on pattern recognition (ICPR'96), p 225
48. Li J, Yang S, Wang X, Xue X, Li B (2009) Tree-structured data regeneration with network coding in distributed storage systems. In: Proceedings of international conference on image processing, Charleston, USA, pp 481–484

49. Kang ANC, T Lee RC, Chang CL, Chang SK (1977) Storage reduction through minimal spanning trees and spanning forests. IEEE Trans Comput C-26:425–434
50. Osteen RE, Lin PP (1974) Picture skeletons based on eccentricities of points of minimum spanning trees. SIAM J Comput 3:23–40
51. Graham RL, Hell P (1985) On the history of the minimum spanning tree problem. IEEE Ann Hist Comput 7(1):43–57
52. Jain A, Mamer JW (1988) Approximations for the random minimal spanning tree with applications to network provisioning. Oper Res 36:575–584
53. Torkestani Akbari J, Meybodi MR (2010) Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs. Int J Uncertain Fuzziness Knowl-Based Syst (to appear)
54. Torkestani Akbari J, Meybodi MR (2010) Mobility-based multicast routing algorithm in wireless mobile ad hoc networks: a learning automata approach. J Comput Commun 33:721–735
55. Torkestani Akbari J, Meybodi MR (2010) A new vertex coloring algorithm based on variable action-set learning automata. J Comput Inf 29(3):1001–1020
56. Torkestani Akbari J, Meybodi MR (Feb. 2010) Weighted steiner connected dominating set and its application to multicast routing in wireless MANETs, Wireless personal communications, Springer, Berlin
57. Torkestani Akbari J, Meybodi MR (2010) An efficient cluster-based cdma/tdma scheme for wireless mobile ad-hoc networks: a learning automata approach. J Netw Comput Appl 33:477–490
58. Torkestani Akbari J, Meybodi MR (2010) Clustering the wireless ad-hoc networks: a distributed learning automata approach. J Parallel Distrib Comput 70:394–405
59. Torkestani Akbari J, Meybodi MR (2010) An intelligent back bone formation algorithm in wireless ad hoc networks based on distributed learning automata. J Comput Netw 54:826–843
60. Billard EA, Lakshmivarahan S (1999) Learning in multi-level games with incomplete information. Part I. IEEE Trans Syst Man Cybern-Part B: Cybern 19:329–339
61. Meybodi MR (1983) Learning automata and its application to priority assignment in a queuing system with unknown characteristics, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Oklahoma, Norman, Oklahoma, USA
62. Hashim AA, Amir S, Mars P (1986) Application of learning automata to data compression In: Narendra KS (ed.) Adaptive and learning systems. Plenum, New York, pp 229–234
63. Oommen BJ, Hansen ER (Aug. 1987) List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations. SIAM J Comput 16:705–716
64. Unsal C, Kachroo P, Bay JS (1999) Multiple stochastic learning automata for vehicle path control in an automated highway system. IEEE Trans Syst Man Cybern-Part A 29:120–128
65. Barto AG, Anandan P (1985) Pattern-recognizing stochastic learning automata. IEEE Trans Syst Man Cybern SMC-15:360–375
66. Akbari Torkestani J, Meybodi MR (2010) A learning automata-based cognitive radio for clustered wireless ad-hoc networks. J. Netw. Syst. Manag. (to appear)