J Supercomput (2013) 64:226–249
DOI 10.1007/s11227-012-0851-1

Degree constrained minimum spanning tree problem: a learning automata approach

Javad Akbari Torkestani

Published online: 9 January 2013

Abstract Degree-constrained minimum spanning tree problem is an NP-hard bicriteria combinatorial optimization problem seeking for the minimum weight spanning tree subject to an additional degree constraint on graph vertices. Due to the NP-hardness of the problem, heuristics are more promising approaches to find a near optimal solution in a reasonable time. This paper proposes a decentralized learning automata-based heuristic called LACT for approximating the DCMST problem. LACT is an iterative algorithm, and at each iteration a degree-constrained spanning tree is randomly constructed. Each vertex selects one of its incident edges and rewards it if its weight is not greater than the minimum weight seen so far and penalizes it otherwise. Therefore, the vertices learn how to locally connect them to the degree-constrained spanning tree through the minimum weight edge subject to the degree constraint. Based on the martingale theorem, the convergence of the proposed algorithm to the optimal solution is proved. Several simulation experiments are performed to examine the performance of the proposed algorithm on well-known Euclidean and non-Euclidean hard-to-solve problem instances. The obtained results are compared with those of best-known algorithms in terms of the solution quality and running time. From the results, it is observed that the proposed algorithm significantly outperforms the existing method.

Keywords Minimum spanning tree · Degree-constrained minimum spanning tree problem · Learning automata

1 Introduction

The constrained minimum spanning tree problem is a combinatorial optimization problem in which two parameters must be optimized at a time. The minimum span-

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

ning tree (MST) of an undirected edge-weighted graph is a spanning tree (SP) with the minimum sum of edge weights among all the spanning trees. The spanning tree of a connected undirected graph is a tree-based sub-graph that connects all the graph vertices. The constrained minimum spanning tree is the spanning tree with the minimum weight subject to an additional constraint. Several forms of the constrained minimum spanning tree problem have been studied in the literature. Bounded diameter minimum spanning tree (BDMST) [32], degree constrained minimum spanning tree (DCMST) [15], capacitated minimum spanning tree (CMST) [33], generalized minimum spanning tree (GMST) [34], delay-constrained minimum spanning tree [35, 36], hop-constrained minimum spanning tree (HMST) [37] are some well-known constrained forms of the minimum spanning tree problem. There exist several well-known algorithms such as Kruskal [38] and Prim [39] that solve the minimum spanning tree problem in polynomial time, where the edge costs are fixed or the search is unconstrained. However, in the case where some additional constraints are imposed to the minimum spanning tree problem, it has been shown [40] that the resulting constrained problem becomes an NP-hard problem.

A spanning tree of a given connected, undirected and edge-weighted graph is said to be degree-constrained if none of whose vertices has a degree greater than $d \geq 2$. Degree-constrained minimum spanning tree problem seeks the degree-constrained spanning tree with the minimum total weight. The DCMST problem in which the spatially distributed components of a system are economically connected under a predefined constraint is an attractive tree structure having many applications in the design of communication systems, computer networks, design of integrated circuits, energy networks, transportation, and so on [5, 18, 30]. By reducing the DCMST problem to an equivalent symmetric traveling salesman problem (TSP), Garey and Johnson [1] showed that this problem is NP-hard. Papadimitriou and Vazirani [2] proved that finding a d-MST (minimum spanning tree with degree constraint d) in the Euclidean plane is known to be NP-hard when $d = 3$. However, Ausiello et al. [3] showed that 3-MST admits a polynomial time approximate solution. Although Papadimitriou and Vazirani [2] conjectured that the DCMST problem is also NP-hard when the degree constraint is 4, Ausiello et al. [3] came to the same conclusion on that 4-MST problem in the Euclidean plane can be solved by a polynomial time approximation solution. Monma and Suri [4] showed that the DCMSP problem can be solved in a polynomial time when $d = 5$. They also showed that there always exists an MST with degree no more than five. Due to the NP-hardness of the DCMST problem, the exact solutions can be only applied to the relatively small problem instances, while very large graphs often arise in real world applications. Therefore, several heuristics such as genetic algorithms (GA) [9, 13, 14, 21, 23, 26, 28], local search [6, 16, 19, 26], ant colony optimization (ACO) [5, 8, 15, 22, 27], particle swarm optimization (PSO) [10, 24, 25, 30], evolutionary computation [11, 29], Lagrangian approach [12, 20, 24], and so on that have shown to be more promising approaches have been proposed to the problem.

Mladenović and Hansen [31] proposed the idea of the variable neighborhood search (VNS). VNS that is mainly based on the local search method explores the state space of the problem by using a dynamic neighborhood technique. Several attempts have been made to extend the VNS method by improving the performance of local search. Souza and Martins [6] proposed an improved version of VNS method

for solving the degree-constrained minimum spanning tree problem. In this work, the authors propose two improvements of the VNS. The former leads the searches to a specific subset of the solution space by a non-random neighborhood selection technique, and the latter one uses the skewed function expressing structural changes in the incumbent solution. The same authors also proposed the VNS-based metaheuristics for solving the min-degree constrained minimum spanning tree problem in [16]. A reduction technique-based algorithm was proposed by Ning et al. [17] for finding a near optimal solution to the DCMST problem. The reduction method reduces the size of the problem and so relieves its hardness. Then, based on the classic Kruskal's algorithm, it uses the edge exchange technique to solve the problem. Knowles and Corne [29] presented a degree constrained minimum spanning tree construction algorithm called RPM (randomized primal method). They applied RPM in three stochastic iterative search methods, simulated annealing, multi-start hill climbing, and genetic algorithm. The results showed that the combination of genetic algorithm and RPM outperforms the others.

Andrade et al. [12] proposed a Lagrangian-based heuristic for DCMST problem. The main objective of the Lagrangian heuristic is to attempt to drive dual solutions into primal feasibility. The Lagrangian relaxation is used to guide the construction of feasible solutions to the problem. Lagrangian relaxation solutions are repeatedly employed to modify the costs for primal heuristics. Ernst [24] designed a hybrid algorithm for the degree-constrained minimum spanning tree problem called CoLaPSO combining the Wedelin's Lagrangian heuristic and particle swarm optimization algorithm. CoLaPSO uses the problem representation that simultaneously works in the dual space (Lagrangian multipliers) and the primal space in the form of the cost perturbations. The hybrid heuristic inherits the capability of calculation of the lower bounds on the objective from the Lagrangian method and the ability to parallelization of algorithm from the PSO. CoLaPSO is one of the baselines with which the results of this paper are compared.

Genetic algorithm has been extensively used in solving the DCMST problem. Zhou and Gen [47] used the Prüfer coding to represent the candidates solutions in GA. Knowles and Corne [29] proposed a GA-based algorithm for the k-MST problem in which the chromosomes are represented as sequences of integer values. Raidl and Julstrom [14] used a weighted coding technique that represents each chromosome with a string of weights associated with the vertices of the target graph.

ACO has been shown to perform well for solving the NP-hard problems. Bau et al. [5] proposed two ACO-based algorithms for solving the DCMST problem called p-ACO and k-ACO. The p-ACO uses the vertices of the construction graph as the solution components and is motivated by the well-known Prim's algorithm for constructing MST, while k-ACO uses the graph edges as the solution components and is based on Kruskal's algorithm. Another ACO-based DCMST algorithm was presented by Doan [22]. Bui et al. [27] presented an improved ACO-based algorithm called AB-DCST to find the low cost degree-constrained spanning trees. In the proposed algorithm, a set of ants moves along the graph and finds the candidate edges that form a degree constrained spanning tree. Then, two local optimization mechanisms are used for further improvements on the constructed spanning tree, 2-EdgeReplacement and 1-EdgeReplacement. The 2-EdgeReplacement algorithm selects two edges from the

spanning tree and replaces them with two new edges in such a way that the degree constraint is met and the tree cost becomes smaller. One edge is selected and replaced by the 1-EdgeReplacement algorithm. AB-DCST is another baseline with which the results of this paper are compared.

In this paper, a decentralized learning automata-based heuristic is presented to find a near optimal solution to the DCMST problem. In this method, each graph vertex is equipped with a learning automaton selecting its incident edges as the actions. This algorithm is composed of several stages and at each stage it constructs a degree-constrained spanning tree at random. To avoid the formation of a cycle, LACT uses variable action-set learning automaton and temporarily removes the actions corresponding to the already selected edges. LACT starts the tree construction process at the root node and each vertex randomly selects one of its incident edges, if any. The selected edge is penalized if its weight is greater than the minimum weight selected by the given vertex so far. It is rewarded otherwise. The vertex at the other end of the selected edge continues the tree construction process and this is repeated until a degree-constrained spanning tree is formed subject to the degree constraint. Each vertex locally decides on the optimality of its selected edges. As the proposed algorithm proceeds, for each vertex, the choice probability of the minimum weight incident edge converges to one. The convergence of the proposed algorithm to the optimal (or ε-optimal) solution is theoretically proved. To show the performance of the proposed algorithm, its results are compared with the best results provided by several recent works reported in the literature [24, 27] in terms of the solution quality and running time. Numerical results confirm the superiority of the proposed algorithm over the existing methods.

The rest of the paper is organized as follows. Section 2 introduces the learning automata theory in a nutshell. In Sect. 3, the new proposal for solving the DCMST problem is presented. Section 4 represents the convergence proof of the proposed DCMST algorithm. Section 5 shows the performance of the proposed algorithm through simulation experiments. Section 6 concludes the paper.

2 Learning automata theory

A learning automaton [41, 42] 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.

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 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] to 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 [41]: 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 the 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 $p(k)$ denote the action selected by learning automaton and the probability vector defined over the action set at an 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 the learning automaton. At each instant k, the action probability vector $p(k)$ is updated by the linear learning algorithm given in Eq. (1), if the selected action $\alpha_i(k)$ is rewarded by the random environment, and it is updated as given in Eq. (2) if the taken action is penalized:

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

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

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. Learning automata have a wide variety of applications in combinatorial optimization problems [52, 56, 57], computer networks [51, 54, 55, 60], Grid computing [48, 50, 59], and Web engineering [49, 53, 58].

2.1 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, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$. $A = \{A_1, A_2, \ldots, A_m\}$ 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 $\Psi_i(k) = \text{prob}[A(k) = A_i \mid A_i \in A,\ 1 \le i \le 2^n - 1]$.

$\hat{p}_i(k) = \text{prob}[\alpha(k) = \alpha_i \mid A(k),\ \alpha_i \in A(k)]$ 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

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

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 Eq. (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 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].

3 DCMST algorithm

Garey and Johnson [1] reduced the DCMST problem to an equivalent symmetric traveling salesman problem and showed that it is NP-hard. Obviously, due to the NP-hardness of the DCMST problem, the exact algorithms become inefficient as the size of the graph instance grows, while very large graphs often arise in real life applications. Since it often suffices to find a near optimal solution in applications, several polynomial time approximation algorithms and heuristics have been proposed. In this paper, a heuristic based on learning automata theory is proposed for approximating a near optimal solution of the well-known degree-constrained minimum spanning tree problem.

3.1 Problem statement

In this section, some preliminaries and definitions are presented to provide a sufficient background to formulate the DCMST problem and to understanding the basics of the proposed algorithm.

Definition 1 Consider a connected, undirected graph $G\langle V, E\rangle$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the graph vertex-set, and $E = \{e_{(v_i,v_j)}\} \subseteq V \times V$ denotes the set of edges. The sub-graph induced by $G'\langle V', E'\rangle$ is a spanning tree of graph $G\langle V, E\rangle$, if the following three conditions are met:

1. Sub-graph G' is connected;
2. $V' = V$ (graphs G and G' have the same vertex-set);
3. $|E'| = |V| - 1$ (where $|x|$ denotes the cardinality of set x).

Definition 2 Consider an edge-weighted, connected, and undirected graph $G\langle V, E, W\rangle$, where V denotes the vertex-set, E denotes the edge-set, and $W = \{w_{e_{(v_i,v_j)}} \mid \forall e_{(v_i,v_j)} \in E\}$ is the set of weights associated with the graph edges. Let

$$\textbf{Step 1.} \quad w_{\tau^l} = \sum_{\forall e_{(v_i,v_j)} \in \tau^l} w_{e_{(v_i,v_j)}} \tag{4}$$

denote the total weight of spanning tree τ^l, where $w_{e_{(v_i,v_j)}}$ denotes the weight of edge $e_{(v_i,v_j)} \in E$. Let $T = \{\tau^1, \tau^2, \tau^3, \ldots\}$ denote the set of all possible spanning trees of graph G. The spanning tree $\tau^* \in T$ is the minimum spanning tree (MST) of graph G, if

$$\textbf{Step 2.} \quad w_{\tau^*} = \min_{\forall \tau^l \in T} \{w_{\tau^l}\}. \tag{5}$$

Definition 3 Consider a connected, undirected, and edge-weighted graph $G\langle V, E, W\rangle$. Let Δ_i denote the degree of vertex $v_i \in V$. The degree of vertex v_i, Δ_i, is defined as the number of vertices adjacent to this vertex (or the number of edges incident at vertex v_i). A degree-constrained spanning tree of graph $G\langle V, E, W\rangle$ is a spanning tree of G subject to

$$\textbf{Step 3.} \quad \Delta_i \leq d \tag{6}$$

for all $v_i \in V$, where d is a positive integer number denoting the degree constraint. Let T_d denote the subset of possible spanning trees of graph G subject to the degree constraint d. The degree-constrained spanning tree $\tau_d^* \in T_d$ is the DCMST of graph G, if

$$\textbf{Step 4.} \quad w_{\tau_d^*} = \min_{\forall \tau_d^l \in T^d} \{w_{\tau_d^l}\}, \tag{7}$$

where

$$\textbf{Step 5.} \quad w_{\tau_d^l} = \sum_{\forall e_{(v_i,v_j)} \in \tau_d^l} w_{e_{(v_i,v_j)}} \tag{8}$$

denotes the total weight of the degree-constrained spanning tree τ_d^i.

3.2 A learning automata-based approach

Let d be the degree constraint, let $G\langle V, E, W\rangle$ denote the connected, undirected, and edge-weighted input graph, and Δ_i be the degree of vertex $v_i \in V$. The proposed algorithm exploits the synergy resulting from the cooperation between a set of connected learning automata the so-called NLA (Network of Learning Automata) to find a near optimal solution to the DCMST problem described in Definition 3. To create the NLA, a learning automaton A_i is associated with each vertex $v_i \in V$. NLA can be defined as a duple $\langle A, \alpha\rangle$, where $A = \{A_i \mid \forall v_i \in V\}$ denotes the set of learning automata, and $\alpha = \{\alpha_i \mid \forall A_i \in A\}$ denotes the set of all actions of NLA. For this problem, NLA is configured as follows:

3.2.1 Configuration of NLA

Let α_i denote the set of all actions that can be taken by the learning automaton A_i associated with vertex v_i. To form an action-set α_i, an action is reserved for selection of every neighbor of vertex v_i (or for every edge with an endpoint at vertex v_i). In other words, an action $\alpha_i^j \in \alpha_i$ is associated with every edge $e_{(v_i,v_j)} \in E$ that connects vertex v_i to one of its adjacent vertices v_j. So, $|\alpha_i| = \Delta_i$, i.e., the cardinality of the action-set α_i equals to the degree of vertex v_i. Selection of action α_i^j by vertex v_i means that this vertex invites vertex v_j to take part in the tree construction process. Selection of α_i^j adds edge $e_{(v_i,v_j)}$ to the spanning tree that is under construction. The network of learning automata that is thus created is isomorphic to the input graph G. Let $p_i = \{p_i^j \mid \forall \alpha_i^j \in \alpha_i\}$ be the action probability vector of the learning automaton A_i, where p_i^j denotes the choice probability of the action α_i^j. At first, the internal state of each learning automaton belonging to NLA is impartially set up in such a way that all actions are chosen with the same probability. To this end, the choice probability of each action α_i^j of the learning automaton A_i is initialized to $\frac{1}{\Delta_i}$, where Δ_i represents the number of neighbors of vertex v_i. To avoid the formation of a cycle in the spanning tree, some actions must be temporarily removed from the action-set of the learning automata. Therefore, the learning automata used in NLA are of variable action-set type as described in Sect. 2.1. The environment in which the automata perform is the P-model and each learning automaton updates its action probability vector by the reinforcement scheme L_{R-I}.

3.2.2 Tree construction process

LACT is an iterative algorithm that randomly constructs a degree-constrained spanning tree at each iteration. In this section, the kth iteration of the proposed algorithm is described. This algorithm starts at a randomly selected vertex v_i (as root vertex). In this algorithm, each learning automaton A_i might be in one of two states: active and inactive. All learning automata are initially set to an active state. If the number of actions selected by each learning automaton A_i equals $d - 1$, its state changes to inactive. This is because no vertex is allowed to have more than d neighbors in DCMST. An inactive learning automaton is not permitted to participate in the tree construction process any longer. Each vertex v_i is connected by edge $e_{(v_i,v_j)}$ to the tree, when it is invited by vertex v_j (or when automaton A_j selects action α_j^i). Therefore, vertex v_i is allowed to select $d - 1$ actions at most. Since the root vertex starts the tree construction process, it is allowed to select d actions.

Let $\tau_d(k)$ denote the spanning tree that is constructed at stage k subject to degree constraint d and let $w_{\tau_d(k)}$ denote the weight of degree-constrained spanning tree $\tau_d(k)$. At each stage k, $\tau_d(k)$ and $w_{\tau_d(k)}$ are initialized to an empty set and zero, respectively. Upon invitation of each vertex v_i, its associated learning automaton A_i prunes its action-set by temporarily disabling the (already selected) actions corresponding to the edges in $\tau_d(k)$. This avoids the formation of a loop in $\tau_d(k)$. Then, the learning automaton A_i checks its action-set α_i to see if it has any more actions (i.e., $\alpha_i \neq \emptyset$). If so, the learning automaton A_i chooses one of its possible actions

at random (say, action α_i^j). The edge corresponding to the chosen action (i.e., edge $e_{(v_i,v_j)}$) is added to $\tau_d(k)$ and the weight of the selected edge is added to $w_{\tau_d(k)}$. Let $N_i(k)$ denote the number of actions that are selected by the learning automaton A_i at stage k. At the beginning of each stage, $N_i(k)$ is set to zero. After selection of each action by the automaton A_i, $N_i(k)$ is incremented by one. If $N_i(k) = d-1$ (or if $N_i(k) = d$ for root vertex v_i), then the learning automaton A_i changes its state to inactive. Otherwise, if the learning automaton A_i has no more possible actions to choose (i.e., $\alpha_i = \emptyset$), it is not able to continue the tree construction process. Under such circumstances, one of the graph vertices that has been already selected must resume the construction process. To do so, the current branch (i.e., the path between current vertex v_i and the root vertex) of the under construction tree $\tau_d(k)$ is traced back from the current vertex v_i toward the root vertex, and the active learning automaton with nonempty action-set that is visited first is selected to continue the tree construction process. In this case, if no active learning automaton with nonempty action-set can be found in backtracking of $\tau_d(k)$ and $|\tau_d(k)| < |V|-1$, the current iteration must be terminated. This case occurs when the construction of a spanning tree of G subject to degree constraint d (by such an order of vertex invitation) is impossible.

Let δ_i be the dynamic weight threshold of vertex v_i. The dynamic threshold δ_i always includes the minimum weight of the edges selected (invited) by vertex v_i so far. For each vertex v_i, at the beginning of algorithm, δ_i is set to a large value. Once the learning automaton A_i selects an action, α_i^j, it compares the weight of the selected action, $w_{e_{(v_i,v_j)}}$, with the dynamic weight threshold δ_i. If $w_{e_{(v_i,v_j)}} \leq \delta_i$, then the learning automaton rewards its selected action by Eq. (1) and updates δ_i to $w_{e_{(v_i,v_j)}}$ (by this, δ_i always keeps the weight of the minimum weight edge its selects), otherwise it penalizes the selected action by Eq. (2). Therefore, as the proposed algorithm LACT proceeds, each vertex v_i converges to its minimum weight incident edge and finally, it selects the minimum edge with a probability close enough to one. The learning automaton A_i then sends an invitation to vertex v_j to take part in the tree construction process. The vertex v_j continues the construction process and repeats the same operations as vertex v_i did. The current iteration k is over if either a spanning tree of the input graph G subject to the degree constraint d is constructed (i.e., $|\tau_d(k)| = |V|-1$) or no active learning automaton with nonempty action-set can be found in $\tau_d(k)$ as described earlier.

Before a new iteration of the proposed algorithm starts, each learning automaton enables its disabled actions again. The algorithm stops if every learning automaton chooses one of its actions with a probability greater than a predefined threshold χ. The degree-constrained spanning tree that is selected in the last iteration is returned as the solution of the DCMST problem provided by the proposed algorithm. By an appropriate choice of the stop threshold χ (as close to one as possible), a desired near optimal solution of the DCMST can be found. Figure 1 shows the pseudo-code of the proposed algorithm.

4 Convergence results

In this section, it is proved that by a proper choice of the learning rate, each learning automaton associated with a graph vertex converges to its optimal action if it updates

Algorithm LACT

Input: Graph $G\langle V, E, W\rangle$, degree constraint d, stop threshold χ
Output: DCMST
Begin Algorithm
Repeat
  Repeat
    A_i updates its action-set to avoid cycle
    If $\alpha_i \neq \emptyset$ **Then**
      A_i randomly chooses one of its actions (say, action α_i^j)
      $\tau_d(k) \leftarrow \tau_d(k) + e_{(v_i, v_j)}$
      $w_{\tau_d(k)} \leftarrow w_{\tau_d(k)} + w_{e_{(v_i, v_j)}}$
      $N_i(k) \leftarrow N_i(k) + 1$
      If $w_{e_{(v_i, v_j)}} \leq \delta_i$ **Then**
        A_i rewards its selected action
        $\delta_i \leftarrow w_{e_{(v_i, v_j)}}$
      Else
        A_i penalizes its selected action
      End If
      If $N_i(k) = d - 1$ **Then** {if $N_i(k) = d$ for root vertex v_i}
        A_i changes its state to inactive
      End If
      $v_i \leftarrow v_j$
    Else
      Trace back $\tau_d(k)$ for an active automaton A_j with nonempty action-set
      $v_i \leftarrow v_j$
    End If
  Until ($|\tau_d(k)| = |V| - 1$) or (active automaton with nonempty action-set not found)
  $k \leftarrow k + 1$
  Each automaton A_i enables its disabled actions
Until each automaton chooses its action with a probability higher than χ
End Algorithm

Fig. 1 Pseudo-code of LACT

its internal state (action probability vector) by the updating rules presented in LACT. That is, by choosing an appropriate learning rate, LACT selects the DCMST with a probability close enough to one.

Theorem 1 *Let $p_i^j(k)$ be the choice probability of the edge (say, $e_{(i,j)}$) having the minimum cost incident at vertex v_i by the learning automaton A_i at stage k. If the vector $\underline{p_i}(k)$ is updated according to LACT, then for every error rate $\varepsilon > 0$, there exists a learning rate $a^*(\varepsilon) \in (0, 1)$ such that for all $a \in (0, a^*)$, we have*

$$\text{Prob}\Big[\lim_{k\to\infty} p_i^j(k) = 1\Big] \geq 1 - \varepsilon. \tag{9}$$

Proof To prove this theorem, it is initially required to demonstrate that the choice probability $p_i^j(k)$ is a submartingale process for large values of k and the changes in the conditional expectation of $p_i^j(k)$ are always non-negative. Therefore, the following lemma must be proved first.

Lemma 1 *The changes in the conditional expectation of* $p_i^j(k)$ *are always non-negative, subject to updating vector* $\underline{p_i}$ *by LACT. That is,* $\Delta p_i^j(k) > 0$.

Proof Define the conditional expectation of $p_i^j(k)$ as

$$\Delta p_i^j(k) = E\big[p_i^j(k+1) \mid \underline{p_i}(k)\big] - p_i^j(k). \tag{10}$$

$\Delta p_i^j(k)$ is rewritten as

$$\Delta p_i^j(k) = a p_i^j(k) \sum_{l \neq j} p_i^l(k)\big[c_i^l(k) - c_i^j(k)\big]. \tag{11}$$

Let S_r^o denote the interior of S_r, where $S_r = \{\underline{p_i}(k) \mid 0 \leq p_i^j(k) \leq 1, \sum_l p_i^l(k) = 1\}$. Therefore, we have $p_i^j(k) \in (0,1)$, for all $\underline{p_i} \in S_r^o$. Theorem 1 assumes that action α_i^j is the optimal action (action with the minimum penalty probability) of the learning automaton A_i. That is, the edge $e_{(v_i,v_j)}$ is the minimum cost edge incident at vertex v_i. Hence, it is concluded that for all $l \neq j$, $c_i^l - c_i^j > 0$, where c_i^j is the final value of penalty probability $c_i^j(k)$, where k is large enough. Using the weak law of large numbers, it can be proved that $c_i^j(k)$ converges to c_i^j, for large values of k (i.e., $\lim_{k\to\infty} \text{Prob}[|c_i^j - c_i^j(k)| > \varepsilon] \to 0$). Therefore, we conclude that for large values of k, the right hand side of Eq. (11) is composed of non-negative quantities, and so we have

$$\Delta p_i^j(k) = a p_i^j(k) \sum_{l \neq j} p_i^l(k)\big[c_i^l(k) - c_i^j(k)\big] \geq 0. \tag{12}$$

This completes the proof of Lemma 1. □

Let $\Gamma_i^j(\underline{p_i})$ denote the convergence probability of LACT to the desired state with initial probability vector $\underline{p_i}$. That is,

$$\Gamma_i^j(\underline{p_i}) = \text{prob}\big[p_i^j(\infty) = 1 \mid \underline{p_i}(0) = \underline{p_i}\big].$$

Let $C(S_r) : S_r \to \Re$ be the state space of all real-valued continuously differentiable functions with bounded derivative defined on S_r, where $\Re$ is the real line. If $\psi(.) \in C(S_r)$, the operator U is defined by

$$U\psi(\underline{p_i}) = E\big[\psi\big(\underline{p_i}(k+1)\big) \mid \underline{p_i}(k) = \underline{p_i}\big], \tag{13}$$

where $E[.]$ represents the mathematical expectation. Operator U is linear and preserves the non-negative functions as the expectation of a non-negative function remains non-negative. That is, $U\psi(\underline{p_i}) \geq 0$ for all $q \in S_r$, if $\psi(\underline{p_i}) \geq 0$. If operator U

is repeatedly applied n times (for all $n > 1$), we have

$$U^{n-1}\psi(\underline{p_i}) = E\big[\psi\big(\underline{p_i}(k+1)\big) \mid \underline{p_i}(k) = \underline{p_i}\big].$$

Function $\psi(\underline{p_i})$ is called super-regular (or sub-regular) if and only if $\psi(\underline{p_i}) \geq U\psi(\underline{p_i})$ (or $\psi(\underline{p_i}) \leq U\psi(\underline{p_i})$), for all $\underline{p_i} \in S_r$. $\Gamma_i(\underline{p_i})$ is the only continuous solution of equation $U\Gamma_i^j = \Gamma_i^j$ with the following boundary conditions:

$$\Gamma_i^j\big(e_i^j\big) = 1, \qquad \Gamma_i^j\big(e_i^l\big) = 0; \quad l \neq j. \tag{14}$$

Define

$$\phi_i^j[x, \underline{p_i}] = \frac{e^{-\frac{xp_i^j}{a}} - 1}{e^{-\frac{x}{a}} - 1}$$

where $x > 0$. $\phi_i^j[x, \underline{p_i}] \in C(S_r)$ and it satisfies the boundary conditions above.

In what follows, it is shown that $\phi_i^j[x, \underline{p_i}]$ is a sub-regular function, thus $\phi_i^j[x, \underline{p_i}]$ qualifies as a lower bound on $\Gamma_i^j(e_i)$. Since the sets of super- and sub-regular functions are closed under addition and multiplication by a positive constant, and if $\phi_i(.)$ is super-regular then $-\phi_i(.)$ is sub-regular, it follows that $\phi_i^j[x, \underline{p_i}]$ is sub-regular if and only if

$$\textbf{Step 6.} \quad \theta_i^j[x, \underline{p_i}] = e^{-\frac{xp_i^j}{a}} \tag{15}$$

is super-regular. We now determine the conditions under which $\theta_i^j[x, \underline{p_i}]$ is super-regular. From the definition of operator U given in Eq. (13), we have

$$U\theta_i^j[x, \underline{p_i}] = E\big[e^{-\frac{xp_i^j(k+1)}{a}} \mid \underline{p_i}\big].$$

If $\theta_i^j[x, \underline{p_i}]$ is super-regular, we have

$$U\theta_i^j[x, \underline{p_i}] - \theta_i^j[x, \underline{p_i}] \leq \left[e^{-\frac{xp_i^j}{a}} p_i^j r_i^j e^{-x(1-p_i^j)} + e^{-\frac{xp_i^j}{a}} \sum_{l \neq j} p_i^l r_i^l e^{xp_i^j}\right] - e^{-\frac{xp_i^j}{a}}$$

where r_i^j denotes the probability of rewarding action α_i^j. Define $V[u]$ as follows:

$$\begin{aligned} V[u] &= \begin{cases} \frac{e^u - 1}{u}; & u \neq 0, \\ 1; & u = 0, \end{cases} \\ U\theta_i^j[x, \underline{p_i}] &- \theta_i^j[x, \underline{p_i}] \\ &\leq -xp_i^j e^{-\frac{xp_i^j}{a}} \left[\big(1 - p_i^j\big) r_i^j V\big[-x\big(1 - p_i^j\big)\big] - \sum_{l \neq j} p_i^l r_i^l V\big[xp_i^j\big]\right] \\ &= -xp_i^j \theta_i^j[x, \underline{p_i}] G_i^j[x, \underline{p_i}] \end{aligned} \tag{16}$$

where $G_i^j[x, \underline{p_i}]$ is defined as

$$\big(1 - p_i^j\big) r_i^j V\big[-x\big(1 - p_i^j\big)\big] - \sum_{l \neq j} p_i^l r_i^l V\big[xp_i^j\big].$$

Therefore, $\theta_i^j[x, \underline{p_i}]$ is super-regular if

$$G_i^j[x, \underline{p_i}] \geq 0 \tag{17}$$

for all $\underline{p_i} \in S_r$. $\theta_i^j[x, \underline{p_i}]$ is super-regular if

$$g_i(x, \underline{p_i}) = \frac{V[-x(1-p_i^j)]}{V[xp_i^j]} \leq \frac{\sum_{l\neq j} p_i^l r_i^l}{(1-p_i^j) r_i^j}. \tag{18}$$

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

$$\sum_{l\neq j} p_i^l \min_{l\neq j} \frac{r_i^l}{r_i^j} \leq \frac{1}{1-p_i^j} \sum_{l\neq j} p_i^l \frac{r_i^l}{r_i^j} \leq p_i^l \max_{l\neq j} \frac{r_i^l}{r_i^j}.$$

Substituting $\sum_{l\neq j} p_i^l$ by $1-p_i^j$ in the above inequality, we have

$$\min_{l\neq j} \frac{r_i^l}{r_i^j} \leq \frac{\sum_{l\neq j} p_i^l \frac{r_i^l}{r_i^j}}{\sum_{l\neq j} p_i^l} \leq \max_{l\neq j} \frac{r_i^l}{r_i^j}.$$

From Eq. (18), it follows that $\theta_i^j[x, \underline{p_i}]$ is super-regular if we have

$$g_i(x, \underline{p_i}) \geq \max_{l\neq j} \frac{r_i^l}{r_i^j}.$$

For further simplification, let us employ logarithms as $\Delta(x, \underline{p_i}) = \ln g_i(x, \underline{p_i})$. We have

Step 7. $$-\int_0^x H'(u)\,du \leq \Delta(x, \underline{p_i}) \leq -\int_{-x}^0 H'(u)\,du,$$

where $H'(u) = \frac{dH(u)}{du}$ and $H(u) = \ln V(u)$.

We have

$$\frac{1}{V[x]} \leq \frac{V[-x(1-p_i^j)]}{V[xp_i^j]} \leq V[-x]$$

and

$$\frac{1}{V[x]} = \max_{l\neq j} \frac{r_i^l}{r_i^j}. \tag{19}$$

Let x^* denote the value of x for which Eq. (19) is true. There exists a value of $x > 0$ under which Eq. (19) is satisfied, if $\frac{r_i^l}{r_i^j}$ is smaller than 1 for all $l \neq j$. By choosing $x = x^*$, Eq. (19) holds true. Consequently, Eq. (17) is true and $\theta_i^j[x, \underline{p_i}]$ is a super-regular function, thus

$$\phi_i^j[x, \underline{p_i}] = \frac{e^{-\frac{xp_i^j}{a}} - 1}{e^{-\frac{x}{a}} - 1}$$

is a sub-regular function satisfying the boundary conditions (14). If $\psi_i(.) \in C(S_r)$ is sub-regular with $\psi(e_i^j) = 1$ and $\psi(e_i^l) = 0$ for $l \neq j$, then $\psi(\underline{p_i}) \leq \Gamma_i^j(\underline{p_i})$ for all $\underline{p_i} \in S_r$. Therefore, it is concluded that

$$\phi_i^j[x, \underline{p_i}] \leq \Gamma_i^j(\underline{p_i}) \leq 1.$$

From the definition of $\phi_i^j[x, \underline{p_i}]$, we see that given any $\varepsilon > 0$ there exists a positive constant $a^* < 1$ such that for all $0 < a \leq a^*$, we have

$$1 - \varepsilon \leq \phi_i^j[x, \underline{p_i}] \leq \Gamma_i^j(\underline{p_i}) \leq 1.$$

Hence, it is concluded that the choice probability of minimum cost edge $e_{(i,j)}$ by learning automaton A_i (i.e., probability p_i^j) converges to 1 for large values of k, and hence the proof of Theorem 1. □

5 Experiments

To show the performance of the proposed DCMST algorithm, several simulation experiments were conducted on several well-known Euclidean and non-Euclidean benchmark graph instances. The Euclidean graph instances that are used in this set of experiments are:

- CRD problem instances are two-dimensional Euclidean graphs in which the coordinates of the graph vertices are generated by using a uniform distribution in a two-dimensional plane. In CRD graph instances, the number of vertices ranges from 30 to 100. The weight of the edge connecting every two graph vertices is defined as the Euclidean distance between the vertices. CRD problem instances were used in [7, 19, 20, 43, 44].
- SYM Euclidean graph instances are very similar to CRD graphs but generating the position of the graph vertices in a higher dimensional Euclidean space. In higher dimensional SYM problems, a vertex is more likely to be connected with more vertices in different dimensions. Therefore, this may increase the probability of violation of degree constraint for each vertex. SYM graphs are adopted from [7, 19, 20]. The size of SYM graph instances is between 30 to 70.
- STR. The structured (STR) problem instances groups the graph vertices in clusters. These graphs are generated by randomly distributing the points in a higher dimensional space grouped together as cluster. The number of vertices of the STR graph instances ranges from 30 to 200. STR benchmarks are generated as follows [7]: One vertex is located at the origin and the remaining vertices all are split into clusters of about the same size. One cluster is located 500 units along each of the major axis of the Euclidean space. Cluster vertices are distributed in a hypercube of size 50 units. The number of dimensions changes from 3 to 7.

Experiments are also performed on the following non-Euclidean graph instances:

- SHRD. These graph instances as the name suggests (structured hard) are very hard-to-solve problems that are generated by assignment of non-Euclidean distances to the graph edges in such a way that the number of optimal solutions is limited [7].

The size of these problem instances is between 15 to 150. SHRD problems are generated as follows: The first vertex is connected to all other vertices by an edge of length l. The second one is connected to all except the first (previous ones) by an edge of length $2l$ and so on. SHRD graph instances are taken from [7].

- Three other non-Euclidean problem instances on which the simulation experiments are conducted are hard-to-solve Random [45], Random-hard (R) [46] and Misleading-hard (M) [29] problems. In Random problem instances, the weight that is associated with each edge is randomly drawn from a predefined interval by using a uniform distribution. The size of Random instances ranges from 15 to 1000. The R and M problem instances are generated with the unconstrained minimum spanning tree to contain star patterns with high degrees. The size of R and M instances is between 50 to 200, and 50 to 500, respectively. M graphs are designed to mislead greedy algorithms [27]. (For more details on non-Euclidean graph generators, the readers are referred to [29].)

The proposed algorithm is tested on the above mentioned problem instances where the degree constraint ranges between 2 and 5, and the obtained results are compared with those of AB-DCST (an ACO-based DCMST algorithm proposed by Bui et al. [27]), CoLaPSO (a hybrid Lagrangian particle swarm optimization algorithm for DCMST problem proposed by Ernst [24]), and the best known methods reported in [27]. To improve the precision of the numerical results, each experiment is repeated 50 times and the results are averaged. Gap is defined to show the difference between the optimal solution (or the solution of the best known method) and the solution provided by the proposed algorithm. Gap is computed as $\frac{|w_A - w_O|}{w_O}$ %, where w_O denotes the weight of the optimal solution or the best known result (if the optimality of the solution has not been proven) and w_A denotes the weight of the solution provided by algorithm A.

Needless to say, the performance of a learning automata-based algorithm is strongly dependent on the choice of a proper learning rate. By choosing an appropriate learning rate, a trade-off between the computational cost of algorithm and its gap to the optimal solution can be made. The cost of algorithm decreases and its gap increases as the learning rate increases, and vice versa. Depending on the application nature, different learning rates can be chosen. If an application sacrifices the cost in favor of the solution quality, a small learning rate is preferred, a larger one can be chosen otherwise. In this paper, we aim to find a learning rate by which a very near to optimal solution can be found to the problem in an as short as possible time. To this end, we tried different learning rates (ranging from 0.05 to 0.5) on all seven types of benchmark graphs and found that the best results are obtained when learning rate is set to 0.09. The stop threshold χ is another parameter that must be precisely tuned in LACT. We tried different values of χ (ranging from 0.30 to 0.95 with increment step 0.05) and found out that the proposed algorithm has the best results for stop threshold 0.9. The learning rate 0.09 and stop threshold 0.9 are used in all further experiments.

The obtained results are summarized in Tables 1–9. In these tables, for each problem instance, the description of the problem including the class of the problem, number of graph vertices and degree constraint are specified. The best previous results

Table 1 Comparison of LACT with the best previous results for CRD problem instances where $d = 2$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
CRD 100	100	7044	7044	20	7033	7.42	0.00156	+
CRD 101	100	7714	7707	23	7701	7.63	0.00078	+
CRD 102	100	7549	7549	22	7549	8.91	0	=
CRD 103	100	7660	7660	22	7651	11.42	0.00117	+
CRD 104	100	7541	7541	19	7537	9.53	0.00053	+
CRD 105	100	7075	7075	21	7075	10.14	0	=
CRD 107	100	7728	7720	18	7698	8.16	0.00285	+
CRD 108	100	7137	7137	19	7138	8.14	−0.0001	−
CRD 109	100	7286	7286	19	7289	7.63	−0.0004	−
CRD 502	50	5480	5480	4.30	5473	1.12	0.00128	+
CRD 503	50	5079	5079	3.34	5069	0.95	0.00197	+
CRD 702	70	6763	6763	11	6755	4.01	0.00118	+
CRD 704	70	6370	6370	7.68	6368	2.76	0.00031	+
CRD 706	70	6671	6671	10	6671	3.09	0	=
CRD 707	70	6467	6467	9.88	6465	4.25	0.00031	+

Table 2 Comparison of LACT with the best previous results for Random problem instances where $d = 2$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
Random 20	20	48	48	0.40	46	0.42	0.04167	+
Random 25	25	56	56	0.82	58	0.53	−0.0357	−
Random 50	50	119	119	4.38	119	1.31	0	=
Random 100	100	246	246	30	238	4.12	0.03252	+
Random 200	200	574	574	111	571	11.20	0.00523	+
Random 300	300	840	840	362	840	25.16	0	=
Random 400	400	1277	1277	375	1256	48.92	0.01644	+
Random 500	500	1706	1706	810	1710	61.22	−0.0023	−
Random 1000	1000	3911	3911	6440	3909	260.41	0.00051	+

are also provided for comparison. In Tables 1–8, the results of the proposed algorithm and AB-DCST [27] are given in terms of the solution value (SV) and running time in seconds (RT). The gap between the results of the proposed algorithm LACT and the best previous results is also given. The last column of Tables 1–9 shows the performance of the proposed method in terms of the quality of the provided solution in comparison with the best previous methods. Symbol '+' in the last column of the tables labeled as 'Performance' means that the proposed algorithm outperforms the best previous methods, symbol '−' implies that the proposed algorithm performs worse, and symbol '=' represents that the results of LACT are as good as those of

Table 3 Comparison of LACT with the best previous results for SHRD problem instances where $d = 2$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
SHRD 1000	100	48145	48141	24	48098	6.49	0.00089	+
SHRD 1500	150	109691	109681	52	109672	9.10	8.20E–05	+

Table 4 Comparison of LACT with the best previous results for STR problem instances where $d = 2$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
STR 502	50	6126	6126	4.54	48098	5.02	0.00098	+
STR 1001	100	5000	5000	22	109672	8.97	0.0018	+
STR 1002	100	7089	7089	23	48098	8.16	−0.0003	−
STR 1003	100	7089	7089	22	109672	7.89	−0.0004	−
STR 1004	100	8699	8699	24	48098	8.23	0.00057	+
STR 1005	100	8699	8699	25	109672	9.11	0.00103	+
STR 1006	100	10529	10529	23	48098	10.07	0	=
STR 1007	100	10529	10529	22	109672	10.41	0.00171	+
STR 1008	100	12428	12428	25	48098	9.65	−0.0002	−
STR 1009	100	12408	12408	21	109672	8.92	0.00073	+
STR 1500	150	14009	14009	62	48098	14.23	0.00086	+
STR 1508	150	13887	13887	65	109672	14.42	0.00036	+
STR 2008	200	15872	15872	95	48098	16.81	0.00057	+

the best known methods. Symbol '$*$' next to a value shows that the optimality of the value has been proven.

Tables 1–5 summarize the obtained results for different classes of the problem instances where degree constraint is set to 2. Tables 6, 7, and 8 show the obtained results for degree constraints 3, 4, and 5, respectively, Table 9 compares the results of LACT with CoLaPSO [24] and AB-DCST [27] where degree constraint changes from 2 to 5, and Table 10 shows the overall performance of LACT as compared to the best-known results in terms of the quality of the provided solution. The obtained results of LACT for CRD problem instances where degree constraint is 2 are given in Table 1. Comparing the results, it is observed that for 10 out of 15 problem instances (66.66 %) LACT finds better results than the best known vales, for 3 cases (20.0 %) it finds the best previous values and only for two remaining instances (13.33 %) it performs worse. The obtained results also show that in all cases the running time of the proposed algorithm is shorter than that of AB-DCST. The same experiments are performed on 9 problem instances of class Random where $d = 2$ and the obtained results are reported in Table 2. Form the results given in this table, it can be seen that in 5 out of 9 instances (55.55 %) the proposed algorithm achieves better results, for 2 cases (22.22 %) the results of LACT is equal to the best known results, and for 2 other problem instances (22.22 %) it has the worst result. For Random problem in-

Table 5 Comparison of LACT with the best previous results for SYM problem instances where $d = 2$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
SYM 502	50	2116*	2150	5.04	2116	2.09	0	=
SYM 503	50	1965*	1991	5.02	1965	2.27	0	=
SYM 506	50	1917*	1932	5.52	1917	2.14	0	=
SYM 508	50	1861	1866	5.44	1857	3.16	0.002149	+
SYM 700	70	2012*	2093	10	2012	3.12	0	=
SYM 701	70	1931	1966	10	1942	2.80	−0.0057	−
SYM 702	70	1864*	1937	9.98	1864	2.91	0	=
SYM 703	70	1497*	1536	10	1501	3.05	−0.00267	−
SYM 704	70	2017	2089	9.70	2014	3.08	0.001487	+
SYM 705	70	2156	2175	10	2155	2.67	0.000464	+
SYM 706	70	1489	1522	11	1489	3.20	0	=
SYM 707	70	2371	2409	10	2389	3.66	−0.00759	−
SYM 708	70	1816	1848	9.88	1812	2.99	0.002203	+
SYM 709	70	1749	1798	10	1748	2.76	0.000572	+

Table 6 Comparison of LACT with the best previous results where $d = 3$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
Random 20	20	36	37	0.32	34	0.37	0.055556	+
Random 25	25	42	43	0.54	42	0.41	0	=
Random 200	200	378	378	111	374	9.43	0.010582	+
Random 300	300	518	518	276	509	21.37	0.017375	+
Random 400	400	723	723	355	716	42.90	0.009682	+
Random 500	500	879	879	563	879	48.10	0	=
Random 1000	1000	1855	1855	3579	1890	182.01	−0.01887	−
SHRD 1500	150	72678	72682	84	72671	7.19	9.63E–05	+

stances, it can be seen that the running time of LACT is significantly shorter than that of AB-DCST (especially for larger graphs) unless for instance 'Random 20'. Table 3 summarizes the obtained results only for two problem instances of class SHRD where degree constraint is 2. The results show the superiority of the proposed algorithm for all cases over the best methods both in terms of the running time and the solution quality.

Table 4 compares the results of LACT for 13 STR problem instances with the best known results where $d = 2$. The results show that for 9 out of 13 problem instances (69.23 %), the results of LACT are superior to the best known results. In 3 cases (23.07 %), LACT does not perform well and the obtained results are worse than those of the best previous methods. Only for 1 out of 13 instances, Gap is zero. Comparing the running time of the proposed algorithm with that of AB-DCST, it can be seen that

Table 7 Comparison of LACT with the best previous results where $d = 4$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
R 50n2	50	4.26	4.26	4.36	4.12	2.09	0.032864	+
R 100n1	100	8.06	8.06	19	8	4.15	0.007444	+
R 100n2	100	8.06	8.06	21	7.98	4.43	0.009926	+
R 100n3	100	8.02	8.02	19	8.02	4.76	0	=
R 200n1	200	16.09	16.09	81	16.01	7.89	0.004972	+
R 200n2	200	15.69	15.69	113	15.71	8.12	−0.00127	−
R 200n3	200	15.63	15.63	103	16.11	8.91	−0.03071	−
Random 500	500	847	847	316	836	35.72	0.012987	+
Random 1000	1000	1774	1774	2808	1774	137.73	0	=
SHRD 1000	100	23649	23652	28	23645	4.93	0.000169	+
SHRD 1500	150	54196	54196	74	54193	6.26	5.54E–05	+

Table 8 Comparison of LACT with the best previous results where $d = 5$

Problem instance		Best results	AB-DCST		LACT		Gap	Performance
Class	$\|V\|$		SV	RT	SV	RT		
M 300n1	300	40.68*	40.68	370	40.7	17.44	−0.00049	−
M 400n1	400	54.69*	54.82	360	54.69	18.93	0	=
M 500n1	500	79.34*	79.38	620	79.34	16.20	0	=
R 50n1	50	4.04	4.04	4.38	4.1	0.98	−0.01485	−
R 50n2	50	3.94	3.94	4.44	3.91	1.21	0.007614	+
R 50n3	50	3.92	3.92	4.94	3.9	0.10	0.005102	+
R 100n1	100	7.51	7.51	23	7.42	5.89	0.011984	+
R 100n2	100	7.58	7.58	21	7.52	5.27	0.007916	+
R 100n3	100	7.5	7.5	22	7.43	4.96	0.009333	+
R 200n1	200	15.06	15.06	105	14.99	14.28	0.004648	+
R 200n2	200	14.65	14.65	88	14.55	10.63	0.006826	+
R 200n3	200	14.59	14.59	115	14.41	15.21	0.012337	+
Random 1000	1000	1760	1760	1684	1760	89.41	0	=
SHRD 1500	150	43098	43105	68	43092	6.03	0.000139	+

the proposed algorithm significantly outperforms AB-DCST except for graph ‘STR 502’. Experiments are repeated for 14 problem instances of class SYM. The obtained results are summarized in Table 5. From the results given in this table, it is clear that LACT outperforms the other methods in 5 out of 14 instances (35.71 %). The results also show that for 5 out of 6 SYM problems with proven optimal values, the proposed algorithm finds the optimal solution of the problem. In all 14 cases of SYM instances, the running time of LACT is considerably shorter than that of AB-DCST.

Table 9 Comparison of LACT with CoLaPSO [24] and AB-DCST [27]

Problem instance			Best results	CoLaPSO	AB-DCST	LACT	Gap	Performance
Class	$\|V\|$	d						
CRD 501	50	2	5564.00	5598.62	5564	5559	0.000899	+
CRD 501	50	3	5130.30*	5130.30	5126	5126	0.000838	+
CRD 700	70	2	6340.00	6357.71	6340	6337	0.000473	+
CRD 700	70	3	5789.55*	5789.55	5789	5789	9.50E–05	+
CRD 100	100	2	7105.00	7173.34	7105	7033	0.01013	+
CRD 100	100	3	6199.05*	6199.05	6196	6196	0.000492	+
SHRD 159	15	2	904.00	904.00	904	903	0.001106	+
SHRD 159	15	3	597.00	597.00	597	595	0.00335	+
SHRD 159	15	4	430.00	430.00	430	431	−0.00233	−
SHRD 159	15	5	332.00	332.00	332	330	0.006024	+
SHRD 200	20	2	1679.00	1679.00	1679	1676	0.001787	+
SHRD 200	20	3	1088.00	1088.00	1088	1089	−0.00092	−
SHRD 200	20	4	802.00	802.00	802	800	0.002494	+
SHRD 200	20	5	627.00	627.00	627	626	0.001595	+
SHRD 259	25	2	2714.00	2714.00	2714	2412	0.111275	+
SHRD 259	25	3	1756.00	1756.00	1756	1755	0.000569	+
SHRD 259	25	4	1292.00	1292.00	1292	1293	−0.00077	−
SHRD 259	25	5	1016.00	1016.00	1016	1012	0.003937	+
SHRD 300	30	2	3992.00	3992.00	3992	3993	−0.00025	−
SHRD 300	30	3	2592.00	2592.00	2592	2587	0.001929	+
SHRD 300	30	4	1905.00	1905.00	1905	1901	0.0021	+
SHRD 300	30	5	1504.00	1506.00	1504	1498	0.003989	+
SYM 500	50	2	1767.74	1767.74	1851	1737	0.017389	+
SYM 500	50	3	1156.00*	1156.00	1156	1156	0	=
SYM 500	50	4	1105.00*	1105.00	1105	1105	0	=
SYM 500	50	5	1098.00*	1098.00	1098	1098	0	=
SYM 701	70	2	1931.00	1931.00	2169	1942	−0.005690	−
SYM 701	70	3	1270.00*	1270.00	1270	1270	0	=
SYM 701	70	4	1198.00*	1198.00	1198	1198	0	=
SYM 701	70	5	1186.00*	1186.00	1186	1186	0	=
STR 100	100	2	5007.00	5018.26	5007	5001	0.001198	+
STR 100	100	3	4702.00*	4702.00	4702	4702	0	=
STR 100	100	4	4546.00*	4546.00	4546	4546	0	=
STR 100	100	5	4403.00*	4403.00	4403	4403	0	=
STR 500	50	2	4420.00	4420.00	4420	4418	0.000452	+
STR 500	50	3	4118.00*	4118.00	4118	4128	−0.00243	−
STR 500	50	4	3956.00*	3956.00	3956	3962	−0.00152	−
STR 500	50	5	3807.00*	3807.00	3807	3807	0	=
STR 700	70	2	4696.00	4700.92	4696	4666	0.006388	+
STR 700	70	3	4397.00*	4397.00	4397	4397	0	=
STR 700	70	4	4245.00*	4245.00	4245	4249	−0.00094	−
STR 700	70	5	4100.00*	4100.00	4100	4100	0	=

Table 10 Overall performance of LACT as compared to the best-known results

Problem instance		Overall performance		
Class	Number of instances	Better	Worse	No difference
CRD	20	75.00 %	10.00 %	15.00 %
M	3	0.00 %	33.33 %	66.67 %
R	16	75.00 %	18.75 %	6.25 %
Random	19	52.63 %	15.79 %	31.58 %
SHRD	22	81.82 %	18.18 %	0.00 %
STR	25	48.00 %	24.00 %	28.00 %
SYM	21	28.57 %	14.29 %	57.14 %
Average performance		51.57 %	19.19 %	29.23 %

The obtained results for degrees 3, 4, and 5 are summarized in Tables 6, 7, and 8, respectively. Table 6 shows that LACT performs better in 5 out of 8 cases (62.5 %) and performs worse only for one problem instance (12.5 %). When LACT degree constraint is set to 4 (in Table 7), LACT outperforms in 7 out of 11 cases (63.63) and produces worse results only in 18.18 % cases. Table 8 compares the results where $d = 5$. This table shows that LACT finds better results in 11 out of 14 cases (78.57 %) and finds the optimal result in 2 out of 3 cases. The results given in Tables 6, 7, and 8 also show that LACT outperforms AB-DCST in terms of the running time in all cases, except for instance 'Random 20'.

Another experiment is conducted to compare the results of the proposed DCMST algorithm with the best results reported in recent works CoLaPSO [24] and AB-DCST [27]. Experimental results are summarized in Table 9. Comparing the results of CoLaPSO and AB-DCST, it can be seen that AB-DCST performs better than CoLaPSO in 9 out of 42 cases (21.42 %), while CoLaPSO performs better only in 2 cases (4.76 %). The results given in Table 9 show that LACT generates significantly better results for 22 out of 42 problem instances (52.38 %). It finds the optimal solution for 12 out of 18 problem instances with the proven optimal values.

Table 10 shows the overall performance of the proposed algorithm in comparison with the best existing results in terms of the solution quality. From the table, it can be seen that LACT performs better for problem instances of class SHRD (81.82 %), CRD (75.00 %) and R (75.00 %). The results also show that the proposed algorithm does not perform well for M (33.33 %) and STR (24.00 %) problem instances. The average results confirm the superiority of the proposed algorithm over the others where it performs better in 51.57 % of instances and finds the best previous results for more than 29 % of the problem instances.

6 Conclusion

In this paper, a learning automata-based decentralized algorithm was proposed for approximating the near optimal solution to the degree-constrained minimum spanning

tree problem that has been shown to be an NP-hard combinatorial optimization problem in graph theory. The proposed algorithm was composed of several stages. At each stage, each vertex rewards its selected edge if its weight is not greater than the minimum weigh seen so far by this vertex. The selected edge is penalized otherwise. As the proposed algorithm proceeds, the choice probability of the optimal and near optimal degree-constrained spanning trees increases while that of the others decreases. The convergence of the proposed method to the optimal or near optimal solution was proved. It was shown that by an appropriate choice of the learning rate the proposed algorithm is able to find the optimal solution with a probability close enough to one. The efficiency of the proposed algorithm was evaluated through conducting extensive simulation experiments on both Euclidean and non-Euclidean problem instances. The obtained results showed that the proposed algorithm considerably outperforms the existing methods both in terms of the solution quality and running time.

References

1. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco
2. Papadimitriou CH, Vazirani UV (1984) On two geometric problems related to the travelling salesman problem. J Algorithms 5(2):231–246
3. Ausiello G, Crescenzi P, Gambosi G, Kann V, Marchetti Spaccamela A, Protasi M (1999) Approximate solution of NP-hard optimization problems. Springer, Berlin
4. Monma C, Suri S (1992) Transitions in geometric minimum spanning trees. Discrete Comput Geom 8:265–293
5. Bau Y-T, Ho C-K, Ewe H-T (2008) Ant colony optimization approaches to the degree-constrained minimum spanning tree problem. J Inf Sci Eng 24(4):1081–1094
6. de Souza MC, Martins P (2008) Skewed VNS enclosing second order algorithm for the degree constrained minimum spanning tree problem. Eur J Oper Res 191:677–690
7. Krishnamoorthy M, Ernst AT (2001) Comparison of algorithms for the degree constrained minimum spanning tree. J Heuristics 7:587–611
8. Bau Y-T, Ho C-K, Ewe H-T (2005) An ant colony optimization approach to the degree-constrained minimum spanning tree problem. In: Computational intelligence and security. Lecture notes in computer science, vol 3801. Springer, Berlin, pp 657–662
9. Soak S-M, Corne D, Ahn B-H (2004) A new encoding for the degree constrained minimum spanning tree problem. In: Knowledge-based intelligent information and engineering systems. Lecture notes in computer science, vol 3213, pp 952–958
10. Binh HTT, Nguyen TB (2008) New particle swarm optimization algorithm for solving degree constrained minimum spanning tree problem. In: Trends in artificial intelligence. Lecture notes in computer science, vol 5351, pp 1077–1085
11. Raidl GR (2000) An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In: Proceedings of the 2000 congress on evolutionary computation, pp 104–111
12. Andrade R, Lucena A, Maculan N (2006) Using Lagrangian dual information to generate degree constrained spanning trees. Discrete Appl Math 154:703–717
13. 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
14. Raidl GR, Julstrom BA (2000) A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In: Proceedings of the 2000 ACM symposium on applied computing, pp 440–445
15. 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
16. Martins P, de Souza MC (2009) VNS and second order heuristics for the min-degree constrained minimum spanning tree problem. Comput Oper Res 36:2969–2982
17. Ning A, Ma L, Xiong X (2008) A new algorithm for degree-constrained minimum spanning tree based on the reduction technique. Prog Nat Sci 18:495–499

18. Kawatra R, Bricker D (2004) Design of a degree-constrained minimal spanning tree with unreliable links and node outage costs. Eur J Oper Res 156:73–82
19. Ribeiro CC, Souza MC (2002) Variable neighborhood search for the degree-constrained minimum spanning tree problem. Discrete Appl Math 118:43–54
20. Volgenant A (1989) A Lagrangean approach to the degree-constrained minimum spanning tree problem. Eur J Oper Res 39:325–331
21. Pereira TL, Carrano EG, Takahashi RHC, Wanner EF, Neto OM (2009) Continuous-space embedding genetic algorithm applied to the degree constrained minimum spanning tree problem. In: Proceedings of IEEE congress on evolutionary computation, pp 1391–1398
22. Doan MN (2007) An effective ant-based algorithm for the degree-constrained minimum spanning tree problem. In: Proceedings of IEEE congress on evolutionary computation, pp 485–491
23. Zhou G, Gen M, Wu T (1996) A new approach to the degree-constrained minimum spanning tree problem using genetic algorithm. In: Proceedings of IEEE international conference on systems, man, and cybernetics, pp 2683–2688
24. Ernst AT (2010) A hybrid Lagrangian particle swarm optimization algorithm for the degree-constrained minimum spanning tree problem. In: Proceedings of IEEE congress on evolutionary computation, pp 1–8
25. Goldbarg EFG, de Souza GR, Goldbarg MC (2006) Particle swarm optimization for the bi-objective degree constrained minimum spanning tree. In: Proceedings of IEEE congress on evolutionary computation, pp 420–427
26. Zeng Y, Wang Y-P (2003) A new genetic algorithm with local search method for degree-constrained minimum spanning tree problem. In: Proceedings of fifth international conference on computational intelligence and multimedia applications, pp 218–222
27. Bui TN, Deng X, Zrncic CM (2012) An improved ant-based algorithm for the degree-constrained minimum spanning tree problem. IEEE Trans Evol Comput 16(2):266–278
28. Han L-X, Wang Y, Guo F-Y (2005) A new genetic algorithm for the degree-constrained minimum spanning tree problem. In: Proceedings of IEEE international workshop on VLSI design and video technology, pp 125–128
29. Knowles J, Corne D (2000) A new evolutionary approach to the degree-constrained minimum spanning tree problem. IEEE Trans Evol Comput 4(2):125–134
30. Guo W-Z, Gao H-L, Chen G-L, Yu L (2009) Particle swarm optimization for the degree-constrained MST problem in WSN topology control. In: Proceedings of international conference on machine learning and cybernetics, pp 1793–1798
31. Mladenović N, Hansen P (1997) Variable neighborhood search. Comput Oper Res 24:1097–1100
32. 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)
33. Oncan T (2007) Design of capacitated minimum spanning tree with uncertain cost and demand parameters. Inf Sci 177:4354–4367
34. Oncan 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
35. 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
36. 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
37. 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
38. Kruskal JB (1956) On the shortest spanning sub tree of a graph and the traveling salesman problem. In: Proceedings of American Mathematical Society, pp 748–750
39. Prim RC (1957) Shortest connection networks and some generalizations. Bell Syst Tech J 36:1389–1401
40. Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. Plenum, New York, pp 85–103
41. Narendra KS, Thathachar KS (1989) Learning automata: an introduction. Prentice-Hall, New York
42. Thathachar MAL, Harita BR (1987) Learning automata with changing number of actions. IEEE Trans Syst Man Cybern 17:1095–1100
43. Narula SC, Ho CA (1980) Degree constrained minimum spanning tree. Comput Oper Res 7:239–249
44. Savelsbergh M, Volgenant T (1985) Edge exchanges in the degree-constrained minimum spanning tree problem. Comput Oper Res 12:341–348

45. Delbem A, de Carvalho A, Policastro C, Pinto A, Honda K, Garcia A (2004) Node-depth encoding for evolutionary algorithms applied to network design. In: Genetic evolutionary computation. Lecture notes in computer science, vol 3102. Springer, Berlin, pp 678–687
46. Boldon B, Deo N, Kumar N (1996) Minimum-weight degree constrained spanning tree problem: heuristics and implementation on an SIMD parallel machine. Parallel Comput 22:369–382
47. Zhou G, Gen M (1998) A note on genetic algorithms for degree constrained spanning tree problems. Networks 30(2):91–95
48. Akbari Torkestani J (2012) A new distributed job scheduling algorithm for grid systems. Cybern Syst (to appear)
49. Akbari Torkestani J (2012) An adaptive learning to rank algorithm: learning automata approach. Decis Support Syst 54(1):574–583
50. Akbari Torkestani J (2012) A distributed resource discovery algorithm for P2P grids. J Netw Comput Appl 35(6):2028–2036
51. Akbari Torkestani J (2012) Backbone formation in wireless sensor networks. Sens Actuators A, Phys 185:117–126
52. Akbari Torkestani J (2012) An adaptive heuristic to the bounded diameter minimum spanning tree problem. Soft Comput 16(11):1977–1988
53. Akbari Torkestani J (2012) An adaptive focused web crawling algorithm based on learning automata. Appl Intell 37(4):586–601
54. J Akbari Torkestani (2012) LAAP: a learning automata-based adaptive polling scheme for clustered wireless ad-hoc networks. Wirel Pers Commun (to appear)
55. Akbari Torkestani J (2012) Mobility prediction in mobile wireless networks. J Netw Comput Appl 35(5):1633–1645
56. Akbari Torkestani J, Meybodi MR (2012) Finding minimum weight connected dominating set in stochastic graph based on learning automata. Inf Sci 200:57–77
57. Akbari Torkestani J (2012) A learning automata-based solution to the bounded diameter minimum spanning tree problem. J Chin Inst Eng (to appear)
58. Akbari Torkestani J (2012) An adaptive learning automata-based ranking function discovery algorithm. J Intell Inf Syst 39(2):441–459
59. Akbari Torkestani J (2012) A new approach to the job scheduling problem in computational grids. Clust Comput 15(3):201–210
60. Akbari Torkestani J (2012) Mobility-based backbone formation in wireless mobile ad-hoc networks. Wirel Pers Commun (to appear)