Journal of Combinatorial Optimization (2020) 40:636–659
https://doi.org/10.1007/s10878-020-00605-0

A rapid learning automata-based approach for generalized minimum spanning tree problem

Masoumeh Zojaji[1] · Mohammad Reza Mollakhalili Meybodi[1] · Kamal Mirzaie[1]

Published online: 13 June 2020

Abstract
Generalized minimum spanning tree problem, which has several real-world applications like telecommunication network designing, is related to combinatorial optimization problems. This problem belongs to the NP-hard class and is a minimum tree on a clustered graph spanning one node from each cluster. Although exact and metaheuristic algorithms have been applied to solve the problems successfully, obtaining an optimal solution using these approaches and other optimization tools has been a challenge. In this paper, an attempt is made to achieve a sub-optimal solution using a network of learning automata (LA). This algorithm assigns an LA to every cluster so that the number of actions is the same as that of nodes in the corresponding cluster. At each iteration, LAs select one node from their clusters. Then, the weight of the constructed generalized spanning tree is considered as a criterion for rewarding or penalizing the selected actions. The experimental results on a set of 20 benchmarks of TSPLIB demonstrate that the proposed approach is significantly faster than the other mentioned algorithms. The results indicate that the new algorithm is competitive in terms of solution quality.

Keywords Generalized minimum spanning tree · Induced subgraph · Learning automata · Combinatorial optimization problems · Random search

1 Introduction

Generalized minimum spanning tree (GMST) was employed to support various problems in the field of graph optimization and has received much attention over the past 30 years (Dror and Haouari 2000). In a clustered graph, GMST aims to find a minimum spanning tree in which every node belongs to only one cluster. Although GMST and the minimum spanning tree (MST) have similar behavior, when the problem is solved

✉ Masoumeh Zojaji
Masoumeh.zojaji@gmail.com

[1] Department of Computer Engineering, Maybod Branch, Islamic Azad University, Maybod, Iran

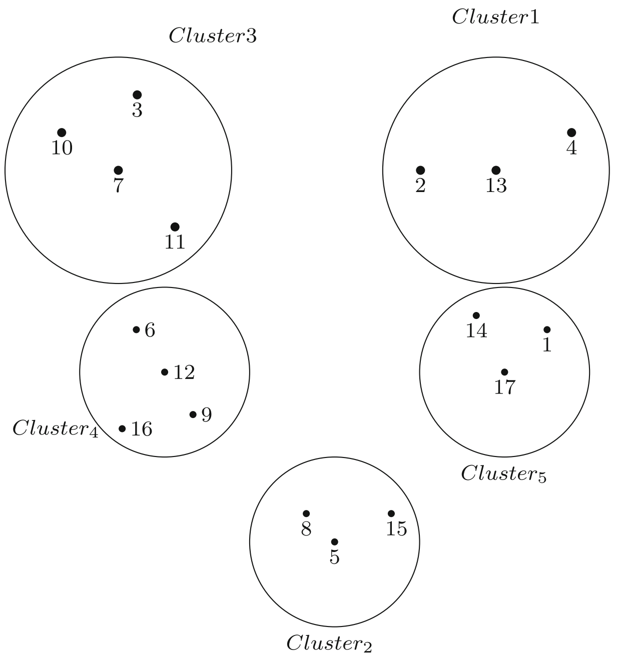


Fig. 1 A clustered graph with 20 nodes

Fig. 2 An array of selected node numbers

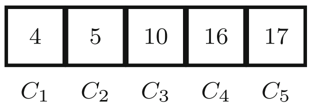
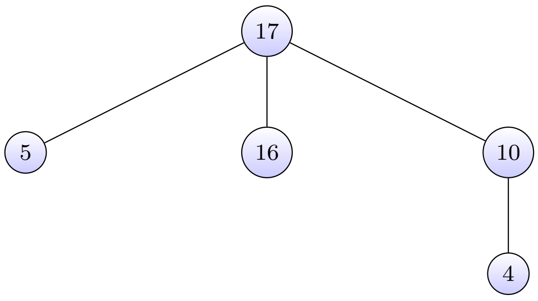


Fig. 3 A generalized spanning tree



by well-known polynomial algorithms of Prim and Kruskal (Golden et al. 2005), the inclusion of the concept of clustering and the selection of a node from each cluster, turns GMST into an NP-hard problem (Myung and Lee 1995). A weighted, undirected graph G in principle can be represented as $G(V, E)$ where V is a set of nodes and E is a set of edges connecting node pairs in V.

In a 'weighted graph', each edge includes additional information called weights and the Euclidean distance between two nodes can be considered as the weight of the corresponding edge. Let R be an equivalence relation partitioning the node-set V into

M equivalent classes called clusters. Figure 1 shows a clustered graph with 20 nodes. After clustering, one node is selected from each cluster (Fig. 2). Next, a spanning tree is constructed over the chosen nodes (Fig. 3). Such a tree is called generalized spanning tree (GST), and the total weight of the GST is equal to the sum of all edges' weight constructing this tree. There might be several GSTs possible. A GMST would be one with the lowest total weight.

In the literature, some exact algorithms have been exposed in an attempt to find an optimum solution for GMST (Pop et al. 2006). These methods are based on various formulations of integer linear programming such as branch and bound, Lagrangian relaxation, and network flow (Haouari and Chaouachi 2006; Pop et al. 2006). However, such approaches are limited by the size of the graph. Consequently, metaheuristic algorithms have been explored to overcome this problem. These are typically problem-independent techniques; therefore, they can be used as black boxes (BoussaïD et al. 2013). In the last 10 years, some metaheuristic methods have been developed to solve the GMST problem.

Specifically, the common drawbacks in these methods are their high execution time and complex parameter tuning. This problem is important in real-time environments. One aspect of environments relevant to GMST problems is cluster head selection in mobile ad-hoc networks (MANETS) (Park et al. 2017; Mukherjee et al. 2018; Dua et al. 2018). An ad-hoc network consists of mobile nodes communicating with each other using wireless links, in the absence of a fixed infrastructure. In the network, the cluster-based topology can be used to reduce the overall network energy consumption (Reddy and Babu 2019; Rao et al. 2017; Prabaharan and Jayashri 2019; Sengottuvelan and Prasath 2017; Farman et al. 2018). In the topology, sensor nodes are grouped into clusters and one node is selected as a cluster head (CH). The selection of an appropriate CH is an optimization problem. This problem can be converted into a GMST problem on a graph when data transmission between clusters is based on a minimum spanning tree algorithm (Tang et al. 2011).

Despite recent researches, solving GMST in a time-sensitive environment is still a major challenge. Therefore, in this paper, a rapid LA-based method is presented to solve GMST.

The proposed algorithm assigns an LA to every cluster, and the number of actions is the same as that of the nodes in the corresponding cluster. At each iteration, each LA selects one node from its cluster. Then, the spanning tree of the selected nodes is constructed. Such a tree is called generalized spanning tree (GST). The total weight of GST is used as a criterion for rewarding or penalizing the selected actions.

Owing to the simple computational complexity of LA and the small number of required parameters, the proposed LA-based method can be applied to time-sensitive environments. The remaining of the paper is outlined as follows: In Sect. 2, some previous metaheuristic-based algorithms are discussed to solve GMST problems. Then, some preliminary materials are defined in Sect. 3. Next, Sect. 4 outlines the fundamentals of the proposed algorithm. Afterward, the experimental results are described in Sect. 5. Finally, Sect. 6 contains conclusions from these experiments.

2 Related work

Myung et al. (Myung and Lee 1995) first introduced the GMST problem, proving that GMST is an NP-hard problem. The problem has been approached by exact methods called integer programming, leading to several mathematical formulations (Pop et al. 2006). Since GMST is an NP-hard problem, it is impossible to solve large instances optimally by exact algorithms. Hence, various metaheuristics have been proposed for this problem.

Golden et al. (Golden et al. 2005) utilized a new genetic algorithm hybridized with a local search using a new operator called tree separation. Their results indicated that both local search and genetic algorithms rapidly generated near-optimal solutions for the GMST problem.

Oncan et al. (Öncan et al. 2008) proposed an efficient tabu search (TS) based method employing innovative neighborhood techniques. Additionally, they developed a new and large data set for the GMST problem called Extended TSPLIB. Their study results demonstrated that the TS-based algorithm yielded the best solutions. Although the performance of the technique compared to a genetic-based algorithm (Golden et al. 2005) is better, it requires long-term memory during execution and high computational time owing to the use of a local search approach.

In Hu's study (Hu et al. 2008), a general variable neighborhood search (VNS) algorithm was proposed to solve the GMST problem by employing three different neighborhood types. Based on their study results, the VNS methodology produced a significantly better solution for the GMST problem and improved other previously proposed metaheuristic algorithms. Furthermore, the algorithm must be investigated to scale up to large problems.

To achieve solutions better than the existing algorithms, Ferreira (Ferreira et al. 2012) proposed a greedy randomized adaptive search procedure (GRASP)-based algorithm, including constructive heuristics and improvement techniques such as path-relinking and iterated local search.

Contreras-Bolton et al. (Contreras-Bolton et al. 2016) introduced a new evolutionary algorithm inspired by the genetic algorithm, named multi-operator genetic algorithm (MOGA) in which two crossover and five mutation operators were utilized for the optimization phase of GMST. MOGA outperforms a mono-operator genetic algorithm. Thus, the use of multi-operators enhances the diversification of the algorithm and allows MOGA to avoid being trapped easily into a local minimum. Although MOGA was an innovative method, it did not scale owing to the high required computational time. The results demonstrate that MOGA outperforms GRASP-based and TS-based algorithms.

Table 1 presents the features of some metaheuristic-based algorithms introduced to solve GMST.

3 Preliminaries

In this section, the useful preliminaries about the GMST problem and LA will be described.

Table 1 A comparison among some features of previously proposed algorithms for solving GMSTP

Feature	References			
	Hu et al. (2008)	Öncan et al. (2008)	Ferreira et al. (2012)	Contreras-Bolton et al. (2016)
Algorithm	Variable Neighborhood Search	Tabu Search	GRASP	Genetic
Mathematic-al model	Local-global MIP formulation	For small instances, linear programming relaxation	For small instances, a model based cut generation	No
Clustering method	Grid clustering	Grid and center clustering	Grid and center clustering	Grid and center clustering
MST Constructing Alg.	Kruskal algorithm	Kruskal algorithm	Prim, Kruskal algorithm	Kruskal algorithm
Benchmark instances	TSPLIB ($137 \leq \|V\| \leq 442$)	TSPLIB ($198 \leq \|V\| \leq 226$),Extended TSPLIB ($229 \leq \|V\| \leq 783$)	TSPLIB ($48 \leq \|V\| \leq 226$),Extended TSPLIB($229 \leq \|V\| \leq 783$)	Extended TSPLIB ($229 \leq \|V\| \leq 783$)
Distance measurement	Euclidean, and non-Euclidean	Euclidean	Euclidean	Euclidean
Compared methods	Genetic,other VNS methods	Genetic	Tabu Search	Tabu Search, GRASP
Pre-processing	No	No	Based on the Bottleneck distance concept	No
Robustness analysis	No	No	Quantile-Quantile plot	No
Stopping conditions	Execution time limit	A balance between computation time and solution quality	For small instances is a target cost or a limit on the execution time for extended TSPLIB is an execution time limit	Predefined maximum number of generations, optimum value or the lower bound
Computation platform	A Pentium IV 2.8 GHz computer with 2 GB RAM	A Pentium IV 3 GHz CPU computer with 2 GB RAM	A Pentium IV 3.2 GHz computer with 2 GB RAM	A Pentium IV 3.2 GHz computer with 2GB RAM
Advantages	Using a Variable Neighborhood Search	Using new neighborhoods solution to improve tabu search	Using the path-relinking iterated local search	Using multiple crossover and mutation operators to solve the GMST that accelerates convergence of the algorithm

3.1 Problem definition

Definition 1 (Graph) A graph can be regarded as a 3-tuple $G = (V(G), E(G), I(G))$ where $V(G) \neq \emptyset$, $V(G) \cap E(G) = \emptyset$, and $I(G)$ is a function which maps an unordered pair (identical or distinct) of $V(G)$ to every member of $E(G)$. The members of $V(G)$ and $E(G)$ are called vertices and edges, respectively. When there is $I_G(e) = \{u, v\}$ for each $e \in E(G)$, denoted by $I_G = uv$, said to be u and v are the endpoints of edge e.

Definition 2 (Subgraph) A subgraph of a graph G, H, is a graph when $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, and $I(H)$ has the same restrictions of $I(G)$ on $E(G)$.

Definition 3 (Vertex-Induced Subgraph) A subgraph of a graph G, H, is called a vertex-induced subgraph if any edge of G whose endpoints are in $V(H)$, is also an edge in H. The induced subgraph of G with the set of vertices $S \subseteq V(G)$ is called the induced subgraph of G by S; it is denoted by $G[S]$.

Definition 4 (Edge-Induced Subgraph) Let $\acute{E}$ and S be a subset of E and V, respectively; let S consist of all endpoints in set $\acute{E}$. Then, the subgraph $(S, \acute{E}, I_G(\acute{e})|\acute{e} \in \acute{E})$ is induced by the set of edges $\acute{E}$ from G; it is denoted by $G[\acute{E}]$.

Definition 5 (Graph with Weighted Edges) If W is a function that assigns a non-negative integer $W(e)$ to every edge $e \in E$, then the resulting graph is called a graph with weighted edges. Let H be a subgraph of G, $W(H)$ denotes the weight of H and equal to the total weights of the edges in this set (Nettleton 2013).

Definition 6 (Generalized Spanning Tree) Let R be an equivalence relation partitioning the vertex set R into m equivalent classes called clusters, where $k = 1, 2, ..., m$ is the set of cluster indices. Then, $V = \bigcup_{i=1}^{m} V_i$ and $\forall_{i \neq f \in k} : V_i \cap V_f = \emptyset$. It is assumed that the edges are defined between pairs of nodes belonging to different clusters. In other words, $\{e_{ij} \in E | i \in V_t, j \in V_s, t \neq s\}$. A tree spanning all clusters and containing a node from each cluster is called a GST. The total weight of the GST(τ) is denoted by $W(\tau) = \sum_{e_{ij} \in \tau} w(i, j)$.

Definition 7 (Generalized Minimum Spanning Tree) Let $T_G = \{\tau_1, \tau_2, ..., \tau_r\}$ be the set of all GSTs of G and $W(\tau_i)$ be the weight of τ_i. Then, the GMST of G is defined by $\tau^* = argmin_{\tau_i} W(\tau_i)$.

In other words, GMST is the generalized spanning tree having the lowest weight among all GSTs. The GMST problem consists of finding a subset of nodes including one node from each cluster. Therefore, the minimum spanning tree of the vertex-induced subgraph $G[S]$ has the lowest weight among all vertex-induced subgraphs.

3.2 Learning automata

Learning automata have long been regarded as a field of artificial intelligence and have been studied extensively in the computer science and engineering problems (Jiang

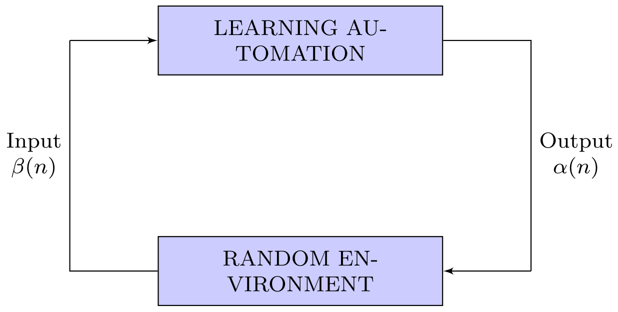


Fig. 4 Relationship between LA and the random environment (Mollakhalili Meybodi and Meybodi 2014)

et al. 2016; Di et al. 2019). For example, covering problem (Rezvanian and Meybodi 2015), wireless network (Mostafaei 2015; Shojafar et al. 2015; Mostafaei et al. 2017; Mostafaei and Meybodi 2013), task offloading in mobile cloud (Krishna et al. 2016), image segmentation (Sang et al. 2016), fuzzy recommender system (Ghavipour and Meybodi 2016), social networks (Ghavipour and Meybodi 2018; Daliri Khomami et al. 2018; Moradabadi and Meybodi 2018; Rezvanian and Meybodi 2017; Moradabadi and Meybodi 2017), spectrum management problem (Fahimi and Ghasemi 2017), autonomous unmanned vehicle (Misra et al. 2017), cloud computing (Ranjbari and Akbari Torkestani 2018; Misra et al. 2014), MST problem (Akbari Torkestani and Meybodi 2011), Cyber-physical system(Ren et al. 2018), clustering (Hasanzadeh-Mofrad and Rezvanian 2018),Wireless mobile Ad-hoc networks (Akbari Torkestani and Meybodi 2010b, a), and vehicular sensor networks (Kumar et al. 2015) are a wide range of research areas for learning automata.

Stochastic learning automata (SLA) as a reinforcement learning method includes a finite set of available actions which is obtained through repeated interactions with a random environment. Therefore, it called adaptive decision-making unit. In this method, how to select an optimal action has an important role in improving the performance of SLA.

At each instant, the automaton chooses an action from its available actions based on a probability distribution over the action set. Indeed, the input to the random environment is the selected action. The environment replies to the action with a reinforcement signal. Therefore, the probability vector of the action is updated by a learning algorithm with the objective of minimizing the average penalty received from the environment. On the other hand, SLA consists of two parts:

1. A stochastic automaton with a limited number of actions and a random environment associated with it.
2. The learning algorithm used by the automata to learn the optimal action.

SLA can be divided into fixed structure SLA (FSLA) and variable structure SLA (VSLA). In the former case, the probability vector of the actions is constant, while in the latter case, the probability vector is updated in each iteration. Figure 4 shows, the relation between VSLA and the random environment. VSLA can be defined by a quadric $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{P}, T, \mathbf{c}\}$ where:

$\boldsymbol{\alpha} \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is a set of actions of the automation/inputs of the environment and $\boldsymbol{\beta} \equiv \{\beta_1, \beta_2, ..., \beta_r\}$ represents a set of inputs of the automation/outputs of the environment; $\mathbf{P} \equiv \{p_1, p_2, ..., p_r\}$ is the probability vector of actions and $T \equiv \mathbf{P} \rightarrow \boldsymbol{\alpha}$ is a learning automata algorithm; $\mathbf{P}$ is the internal state of VSLA which can be considered by the probability vector. A set of penalized probabilities, $\mathbf{c} \equiv \{c_1, c_2, ..., c_r\}$, represents the environment (Mollakhalili Meybodi and Meybodi 2014).

The role of the learning algorithm is to modify the probability vector of the actions in VSLA. The algorithm, which has a significant impact on the performance of VSLA, is explained as follows:

1. If VSLA selects an action $\alpha_i(t)$ from the action set in time t and receives a desired response, then the probability vector of the action $p_i(t)$ is increased and the probability vector of other actions is decreased.
2. If VSLA selects an action $\alpha_i(t)$ and receives an undesired response, then the probability vector of the action $p_i(t)$ is decreased and, the probability vector of other actions is increased. Therefore, the probabilities change as follows:

$$p_{j=1,...,r}(t+1) = \begin{cases} (1-a)\,p_j(t) + a & j = i \\ (1-a)\,p_j(t) & \forall j \neq i \end{cases} \tag{1}$$

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

Where a is a reward parameter, b is a penalty parameter, and r is the number of actions that can be taken by VSLA. Learning algorithms can be divided into three classes based on the relative values of the learning parameters (namely a and b):

- linear reward-penalty method(L_{R-P}): $a = b$.
- linear reward-penalty method($L_{R-\epsilon P}$):$a \gg b$.
- linear reward-inaction(L_{R-I}): $b = 0$. This strategy means the probability vector remains unchanged when the taken action is penalized by the environment.

4 Proposed algorithm

In all cases of the GMST problem, there is a preprocessing phase in which the graph is partitioned into a set of meaningful sub-classes called clusters. In the proposed algorithm, the partitional clustering (Tarabalka et al. 2009) method is considered for the clustering phase, including center clustering and grid clustering, which commonly exist in GMST (Golden et al. 2005; Öncan et al. 2008; Contreras-Bolton et al. 2016). Algorithm 1 and 2 outline both approaches, respectively.

Figure 5 illustrates an example of a grid clustering approach. After the clustering procedure, the optimization phase must be performed to solve the GMST problem. In this article, we propose a novel strategy based on LA to obtain a near-optimal GMST in a graph. Algorithm 3 presents the pseudo-code of the LA-based proposed algorithm.

Algorithm 1: Center clustering procedure

```
Input:
  • $V$: A set of nodes; $|V| = n$.
  • $K_{max}$: An upper bound of the number of clusters; $K_{max} = \lceil n/5 \rceil$
Output: A set of non-empty clusters; $C_1, C_2, ..., C_k$
1  Let $k = K_{max}$.
2  begin
3      Choose $k$ nodes from the set $V$ randomly to serve as cluster centers $C_1, C_2, ..., C_k$.
4      Assign the remaining nodes to the clusters on the basis of the nearest Euclidean distance to the
       cluster center.
5      Eliminate cluster $C_k$ if includes less than one point.
6  end
```

Algorithm 2: Grid clustering procedure

```
Input:
  • $V$: A set of nodes with the known geographical coordinates; $V = \{(x_i, y_i)|i = 1, ..., n\}$.
  • $g$: An integer number used to create a grid; $g \gg n/\mu$.
  • $\mu$: A parameter for tuning the number of the non-empty clusters; It is defined experimentally.
Output: A set of non-empty clusters;$C_1, C_2, ..., C_k$
1  Let $x_{min}, y_{min}$ and $x_{max}, y_{max}$ denote the minimum and maximum coordinates of the $x$ and $y$ ,
   respectively.
2  Let $R$ be a rectangle bounded $x_{min}, y_{min}$ and $x_{max}, y_{max}$.
3  begin
4      Subdivide $R$ into $g \times g$ grids with coordinates.
5      Consider each cell of grid as a cluster $C_1, C_2, ..., C_{g \times g}$.
6      foreach $v \in V$ do
7          Assign $v$ to $C_{g \times g}$ which is bounded to that.
8      end
9      Eliminate cluster $C_{g \times g}$ if it includes less than one point.
10 end
```

Each iteration of the proposed algorithm embodies three steps. The following parts describes the steps of one iteration.

- Step1. After associating an LA with each cluster, one action is selected from the action sets of the corresponding cluster. Every LA and its corresponding cluster have the same number of members. The applied selection mechanism is similar to the roulette wheel selection depending on the probability vector of LA. In Line 13, the notation α^i_{Rnd} means that action with index Rnd is selected from i^{th} cluster. The chosen actions determine a subset of nodes, each node of which is picked only from one cluster $\{S|S \subset V\}$ (Lines 16,17). Figure 6, represents this subset.
- Step2. A spanning tree is derived on the vertex-induced subgraph G_s called GST. Then, a MST is constructed by a greedy algorithm like Kruskal (Golden et al. 2005). Next, the weight of MST, which is equal to the total weight of edges of this tree, is computed (Lines 18,19).
- Step3. The weight of the pre-constructed MST is considered a criterion for the reward or penalty of the actions taken by the automata. Let w_k denote the weight of MST in the k^{th} iteration of the algorithm. If w_k is less than a defined threshold,

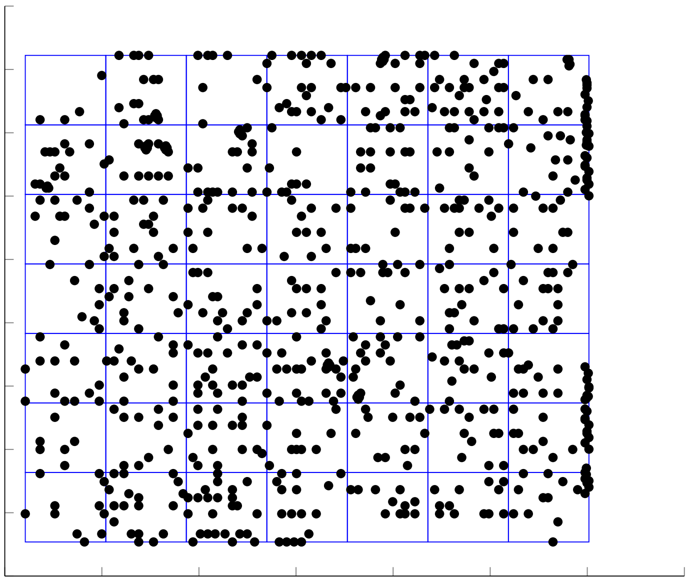


Fig. 5 An example of a grid clustering method

the subset of chosen vertices is rewarded in the first step, otherwise it is penalized (Lines 20–24).

Afterward, the threshold value is updated based on the weight of the resulting MST in the previous step. Let TH_k represent the threshold value in the k^{th} iteration of the algorithm (Line 25). Then, TH_k is updated according to the following formula:

$$\begin{aligned} TH_k &= \frac{(k-1)TH_{k-1} + w_k}{k} \\ &= TH_{k-1} + \frac{w_k - TH_{k-1}}{k} \end{aligned} \tag{3}$$

Finally, the LA-GMST algorithm performs the above steps until a specified number of iterations is exceeded, or alternatively the probability vector of the selected GMST reaches a predefined value (near to one). After stopping the algorithm, the last MST is introduced as GMST (Line 27).

5 Experimental result and discussion

This section aims to analyze the behavior of LA-GMST experimentally. In the LA-GMST algorithm, for stopping criteria, the number of iterations and the threshold of probability are set 10, 000 and 0.95, respectively. The results of the LA-GMST algorithm are compared to those of two famous metaheuristic-based algorithms, including MOGA (Contreras-Bolton et al. 2016) and TS (Öncan et al. 2008). These algorithms have been tested on a standard benchmark instance-sets (extended TSPLIB) including

Algorithm 3: LA-based proposed algorithm(LA-GMST)

```
Input:
  • $G$: A graph: $G = (V, E, W)$; $|V| = n$.
  • $\acute{V}$: Disjointed clusters; $\acute{V} = \bigcup_{i=1}^{m} V_i | \forall_{i,j}$and $V_i \cap V_j = \emptyset, i \neq j$
  • $I_{max}$: Maximum of iteration.
  • $P_{max}$: Maximum of probability.
Output: GMST: A generalized minimum spanning tree with near-optimal weight on graph $G$.
1  Let $N_i$ be the size of cluster $V_i$;$|V_i| = N_i$,$n = \sum_{i=1}^{m} N_i$.
2  begin
3      Let $TH_k$ be the dynamic threshold at stage $k$ and initially set to $\infty$.
4      Let $k$ denotes the stage number and is initially set to 0.
5      Assign a learning automata $LA_i$ to each cluster $V_i$.
6      Let $_i = \{\alpha_1^i, \alpha_2^i, ..., \alpha_{N_i}^i\}$ and $\mathbf{P}_i = \{p_1^i, p_2^i, ..., p_{N_i}^i\}$ denote the action-set and
       action-probability of learning automata $LA_i$ and initially $\forall j \in \{1..N_i\}$  $p_j^i \leftarrow \frac{1}{N_i}$.
7      Let $mst_k$ denotes the minimum spanning tree constructed at stage $k$ and is initially set to $null$.
8      Let $w_k$ denotes the weight of the minimum spanning tree constructed and is initially set to $\infty$.
9      Let $G_S$ denotes a vertex induced subgraph of $G$ at stage $k$, and is initially set to $null$.
10     while $p(k) < P_{max}$ and $k < I_{max}$ do
11         $k \leftarrow k + 1$ and $p(k) \leftarrow 1$.
12         forall the the clusters $V_i(G)$ in parallel do
13             $LA_i$ selects randomly one of its action $(\alpha_{Rnd}^i)$.
14             $p(k) \leftarrow p(k) \times p_{Rnd}^i$.
15         end
16         Selected actions define subset $S = \bigcup_{i=1}^{m} V_i(\alpha_{Rnd}^i)$.
17         According to $S$, vertex induced sub-graph is constructed $G_S$.
18         The minimum spanning tree of $G_S$ is constructed $(mst_k)$.
19         The weight of $mst_k$ is computed by $w_k = \sum_{\forall e_{ij} \in mst_k} w(e_{ij})$.
20         if $w_k \leq TH_{k-1}$ then
21             The chosen actions by $LA_i$, $i = 1, ..., m$ are rewarded.
22         else
23             The chosen actions by$LA_i$, $i = 1, ..., m$ are penalized.
24         end
25         $TH_k \leftarrow TH_{k-1} + \frac{w_k - TH_{k-1}}{k}$
26     end
27     The last induced subgraph $(G_S)$ represents the $GST$ and the last minimum spanning tree $(mst_k)$
       denotes the $GMST$ of the $G$.
28 end
```

the twenty large-scale instances described in (Öncan et al. 2008). Each of the instances is solved by any of LA-GMST, MOGA and TS for 35 times. The Euclidean distance among nodes has been used as the weight of edges. However, comparison of the running times among various metaheuristics is difficult owning to different computing environments. Therefore, to ensure a fair comparison to other algorithms, the same platform is employed. The LA-GMST algorithm and the compared metaheuristics are implemented in MATLAB2016a, and the computational platform is based on PC running on a 2.70 GHz Intel Ci5 processor and 8 GB RAM.

– Experiment 1. In the proposed LA-based algorithm, the parameter a (namely learning rate) should be chosen correctly. Depending on the type of problem, a

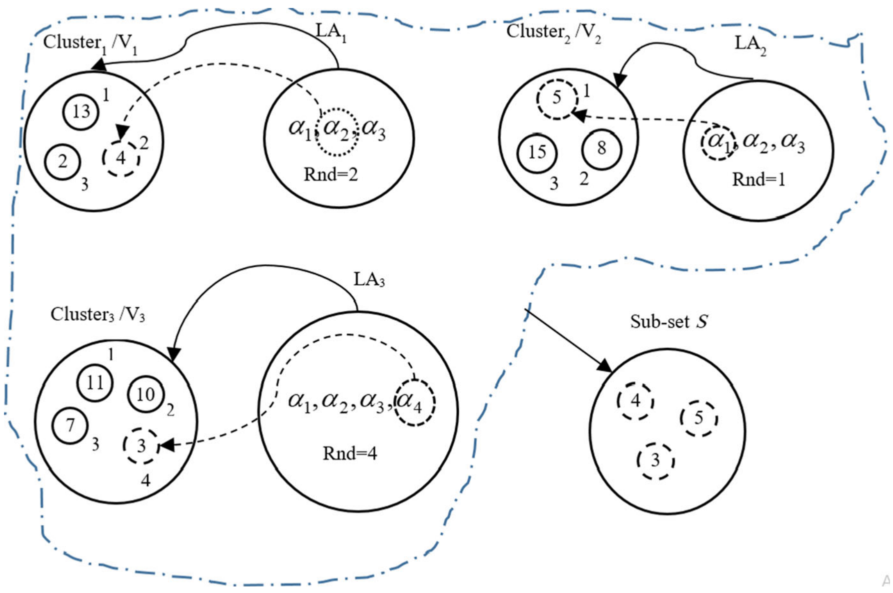


Fig. 6 An example of the vertex-induced subgraph

tuning stage might be needed for this parameter. In this experiment, the learning rate decreases linearly from 0.1 to 0.008.

Table 2 presents the results of the proposed algorithm using four different learning rates. The first column consists of two parts, namely the graph name from TSPLIB benchmarks and the number of nodes. Moreover, the second column denotes the number of clusters. For each learning rate, two columns are considered as $Avg.w$ and $Avg.t$. The columns denote the average weight of the obtained GMST (based on Euclidean distance) and the average CPU time(in seconds), respectively. Meanwhile, the best results are shown in bold. As Table 2 shows, the learning rates 0.1 and 0.06 converge faster than the other two in all of the graph instances. Thereby, the algorithm falls into the trap of the local minimum. It seems that the learning rate of 0.01 and 0.008 have similar performance in some instances. To evaluate the results above, a statistical analysis of the proposed algorithm is investigated for four learning rates.

Table 3 compares the ranking of the learning rates in multi problems. The rankings are obtained based on a non-parametric test (namely Friedman's test). Owning to the dissimilarities in the results and the small size of the analyzed sample, the parametric test may result in unpredictable conclusions (García et al. 2009). As Table 3 shows, it can be observed that the average rank of $a = 0.01$ is smaller than other learning rates.

The p value (0.000) obtained from Friedman's test indicates the existence of a significant difference among results. For this reason, a two-tailed Wilcoxon rank-sum test for the learning rates compared is conducted. The results confirm that the performance of the proposed algorithm with the learning rate 0.01 is statistically better than learning rates 0.1 ($p = 0.000$), 0.06 ($p = 0.000$), and 0.008 ($p = 0.049$), since the p values were all zero at the $\alpha = 0.05$ level. Thus, $a = 0.01$ is used in all experiments reported in the remainder of this section.

– Experiment 2. After tuning the parameters, the effects of clustering on the LA-GMST algorithm are assessed with the selected learning rate and some performance measures. As previously mentioned, two ways of clustering exist for the GMST problem. One possibility is to select k center nodes to be located with each other as far as possible. Another possibility is to determine the $g \times g$ grids in which clusters are constructed based on these grids, and a parameter called μ. Similar to the kinds of literature on GMST, it takes their values in {3, 5, 7, 10} in grid clustering. To make an accurate comparison, the best available weight of GMST in the unlimited computational time was recorded.
Tables 4, 5, 6, 7 and 8 present the results of running the proposed method and other metaheuristics for different clustering modes. Results are averaged over 35 runs. In these tables, for all algorithms, two columns need to be considered including the average weight of the obtained GMST (Ave.weight), and the average CPU time value in seconds (CPU Time).
As Table 4 shows, in all cases, the performance of the LA-GMST algorithm is better than that of TS. Compared to MOGA, LA-GMST outperforms for 16 instances of 20 instances, which are shown in bold. For the average computation time, the proposed algorithm is faster than others.
The TS-based algorithm fails to achieve the best results for different values of μ. When LA-GMST and the MOGA algorithms are compared to each other in terms of the best weights of GMST, $LA{-}GMST_{\mu=3}$ finds 9 best values, $LA{-}GMST_{\mu=5}$ finds 10, $LA - GMST_{\mu=7}$ finds 13, and $LA - GMST_{\mu=10}$ obtains 13. As the results report, it appears that the performance of the proposed algorithm with MOGA is competitive. However, it is worth mentioning that the running time of LA-GMST is considerably shorter than that of MOGA.
To identify statistical differences among compared algorithms, the two-tailed Wilcoxon rank-sum test is conducted at a significant level of $\alpha = 0.05$. Table 9 summarizes the results of the pairwise comparisons of algorithms. The p values indicate that there is either no difference between the weight of the GMST obtained by the first algorithm and the weight of the GMST obtained by the second algorithm (null hypothesis H_0) or the weight of the GMST obtained by the first algorithm better than the weight of the GMST obtained by the second algorithm (alternative hypothesis H_1).

Table 9 confirms that the difference between the LA-GMST algorithm and the TS-based algorithm is statistically significant since the p values were all zero at the $\alpha = 0.05$ level. When the proposed and MOGA algorithms are compared to each other in terms of the p values, LA-GMST significantly yielded better results than MOGA did for instances in generating in the center clustering and grid clustering with $\mu = 10$. While for the grid clustering with $\mu = 3$, $\mu = 5$, and $\mu = 7$, there is no difference between the weight of the GMST obtained by the proposed algorithm and the weight of the GMST obtained by the MOGA algorithm, since

Table 2 Performance comparison of different learning rates

Graph name	K[1]	$a = 0.1$		$a = 0.06$		$a = 0.01$		$a = 0.008$	
		Avg. w[2]	Avg. t	Avg. w[2]	Avg. t	Avg. w[2]	Avg. t	Avg. w[2]	Avg. t
gr229	46	656 (10)	1	641 (8)	3	620 (3)	19	**619 (7)**	19
gil262	53	842 (24)	1	807 (18)	4	**742 (29)**	23	752 (46)	23
pr264	53	22154 (289)	1	21735 (289)	3	**20422 (152)**	23	20506 (435)	23
pr299	60	21242 (342)	2	20734 (288)	5	19647 (137)	28	**19609 (108)**	28
lin318	64	18694 (334)	2	18282 (219)	6	17247 (118)	32	**17232 (207)**	32
rd400	80	5807 (117)	4	5633 (88)	9	**5285 (97)**	53	5341 (188)	53
fl417	84	6463 (55)	4	6412 (41)	10	6307 (29)	58	**6302 (14)**	62
gr431	87	745 (13)	4	731 (10)	10	700 (13)	63	**696 (8)**	74
pr439	88	48481 (585)	4	47937 (601)	11	**45658 (836)**	64	46047 (1228)	67
pcb442	89	20860 (338)	5	20501 (318)	11	**19276 (465)**	65	19391 (557)	69
d493	99	15208 (191)	5	15068 (143)	14	**14511 (191)**	84	14520 (181)	84
att532	107	36499 (529)	7	35825 (427)	16	34328 (884)	96	**34298 (767)**	101
ali535	107	744 (14)	6	722 (14)	17	**675 (15)**	98	**675 (12)**	98
u574	115	15830 (222)	8	15609 (148)	19	**14781 (243)**	119	14787 (247)	110
rat575	115	2504 (35)	8	2441 (32)	21	**2274 (55)**	114	2287 (53)	110
p654	131	17852 (117)	10	17702 (84)	25	**17376 (44)**	147	17381 (83)	144
d657	132	19878 (254)	10	19539 (263)	26	**18674 (393)**	150	18699 (450)	146
gr666	134	1161 (17)	11	1138 (12)	26	**1080 (9)**	156	1081 (20)	151
u724	145	16862 (198)	15	16552 (148)	35	**15744 (186)**	191	15832 (394)	178
rat783	157	3524 (40)	16	3439 (45)	43	**3268 (88)**	214	3294 (108)	221

[1] The number of clusters
[2] Average weight(Standard Deviation)

Table 3 The ranking of comparing different learning rates

Graph name	K[1]	$a = 0.1$	$a = 0.06$	$a = 0.01$	$a = 0.008$
gr229	46	4	3	2	**1**
gil262	53	4	3	**1**	2
pr264	53	4	3	**1**	2
pr299	60	4	3	2	**1**
lin318	64	4	3	2	**1**
rd400	80	4	3	**1**	2
fl417	84	4	3	2	**1**
gr431	87	4	3	2	**1**
pr439	88	4	3	**1**	2
pcb442	89	4	3	**1**	2
d493	99	4	3	**1**	2
att532	107	4	3	2	**1**
ali535	107	4	3	**1.5**	**1.5**
u574	115	4	3	**1**	2
rat575	115	4	3	**1**	2
p654	131	4	3	**1**	2
d657	132	4	3	**1**	2
gr666	134	4	3	**1**	2
u724	145	4	3	**1**	2
rat783	157	4	3	**1**	2
Average rank		4	3	**1.4**	1.6

[1] The number of clusters

Table 4 Computational results for the proposed method, TS, and MOGA for TSPLIB instances with center clustering

Graph name	K[1]	Proposed LA-based		TS (Öncan et al. 2008)		MOGA (Contreras-Bolton et al. 2016)	
		Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]
gr229	46	**620** (**3**)	19 (0)	635 (5)	36 (0)	628 (5)	33 (3)
gil262	53	**742** (**29**)	23 (0)	816 (20)	44 (0)	795 (17)	94 (6)
pr264	53	**20422** (**152**)	23 (0)	21545 (336)	45 (1)	21293 (156)	94 (5)
pr299	60	**19647** (**137**)	28 (0)	20724 (294)	54 (0)	20410 (194)	135 (9)
lin318	64	**17247** (**118**)	32 (1)	18447 (270)	60 (1)	17765 (257)	152 (9)
rd400	80	**5285** (**97**)	53 (0)	5641 (109)	100 (3)	5556 (76)	309 (23)
fl417	84	**6307** (**29**)	58 (0)	6407 (34)	106 (1)	6361 (25)	342 (21)
gr431	87	**700** (**13**)	63 (0)	728 (10)	115 (1)	704 (10)	358 (24)
pr439	88	**45658** (**836**)	64 (1)	47528 (529)	114 (1)	46515 (654)	383 (30)
pcb442	89	19276 (465)	65 (0)	19837 (280)	116 (1)	**19011** (**207**)	401 (26)
d493	99	14511 (191)	84 (5)	14861 (125)	158 (3)	**14374** (**126**)	541 (28)
att532	107	**34328** (**884**)	96 (1)	36461 (556)	187 (4)	35428 (425)	681 (47)
ali535	107	**675** (**15**)	98 (7)	721 (15)	180 (5)	698 (9)	733 (48)
u574	115	**14781** (**243**)	119 (7)	15420 (205)	208 (7)	15024 (232)	842 (53)
rat575	115	**2274** (**55**)	114 (1)	2440 (37)	200 (1)	2323 (63)	940 (547)
p654	131	**17376** (**44**)	147 (1)	17615 (48)	259 (2)	17377 (70)	1240 (80)
d657	132	**18674** (**393**)	150 (1)	19647 (226)	276 (10)	19185 (287)	1249 (101)
gr666	134	1080 (9)	156 (3)	1123 (11)	282 (11)	**1055** (**9**)	1300 (97)
u724	145	15744 (186)	191 (14)	16104 (141)	329 (11)	**15450** (**103**)	1741 (143)
rat783	157	**3268** (**88**)	214 (2)	3479 (30)	397 (9)	3375 (56)	2151 (144)

[1] The number of clusters
[2] Average weight (Standard Deviation)
[3] Average running time (Standard Deviation)

Table 5 Computational results for the proposed method, TS, and MOGA for TSPLIB instances with the grid clustering $\mu = 3$

Graph name	K[1]	Proposed LA-based		TS (Öncan et al. 2008)		MOGA (Contreras-Bolton et al. 2016)	
		Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]
gr229	81	**909 (6)**	40 (1)	935 (6)	36 (1)	921 (5)	148 (8)
gil262	95	**1294 (6)**	55 (1)	1355 (12)	53 (2)	1321 (11)	223 (10)
pr264	101	29307 (96)	65 (4)	29517 (112)	51 (1)	**29251 (23)**	261 (18)
pr299	102	**24107 (395)**	67 (4)	25035 (299)	66 (1)	24758 (205)	280 (14)
lin318	108	**24550 (92)**	78 (1)	25784 (302)	73 (1)	25173 (168)	354 (23)
rd400	135	**7981 (106)**	119 (5)	8391 (67)	129 (2)	8214 (84)	729 (49)
fl417	142	8329 (10)	128 (1)	8318 (6)	127 (2)	**8310 (3)**	801 (48)
gr431	149	1201 (7)	141 (1)	1211 (4)	133 (5)	**1198 (3)**	961 (50)
pr439	163	**66685 (528)**	170 (2)	68222 (383)	161 (7)	67539 (460)	1117 (76)
pcb442	156	**28579 (314)**	155 (1)	29737 (223)	168 (6)	29308 (234)	1055 (82)
d493	171	20674 (50)	191 (1)	20997 (116)	187 (4)	**20665 (118)**	1402 (103)
att532	182	**50953 (436)**	227 (12)	52407 (306)	229 (13)	51581 (299)	1798 (130)
ali535	181	1325 (5)	232 (11)	1340 (4)	239 (9)	**1319 (2)**	1799 (124)
u574	198	20740 (331)	270 (7)	21473 (151)	286 (13)	**20631 (236)**	2262 (139)
rat575	196	3223 (86)	286 (19)	3355 (25)	311 (13)	**3141 (49)**	2281 (127)
p654	177	22994 (34)	237 (11)	22950 (12)	295 (12)	**22916 (6)**	2362 (140)
d657	221	**27343 (390)**	372 (23)	28269 (173)	367 (7)	27496 (534)	3332 (233)
gr666	224	1869 (17)	349 (4)	1901 (7)	360 (17)	**1866 (12)**	3490 (210)
u724	266	23371 (274)	496 (2)	23929 (141)	511 (15)	**22561 (71)**	5427 (349)
rat783	285	4682 (96)	611 (34)	4839 (35)	609 (9)	**4564 (81)**	6741 (522)

[1] The number of clusters
[2] Average weight (Standard Deviation)
[3] Average running time (Standard Deviation)

Table 6 Computational results for the proposed method, TS, and MOGA for TSPLIB instances with the grid clustering $\mu = 5$

Graph name	K[1]	Proposed LA-based		TS (Öncan et al. 2008)		MOGA (Contreras-Bolton et al. 2016)	
		Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]
gr229	47	710 (7)	20 (1)	722 (8)	34 (1)	**706 (4)**	76 (5)
gil262	63	**1020 (9)**	33 (0)	1079 (10)	46 (1)	1042 (14)	121 (8)
pr264	55	**21821 (141)**	27 (0)	23033 (286)	44 (0)	22540 (211)	102 (5)
pr299	69	**19771 (197)**	39 (0)	20455 (300)	56 (1)	20089 (255)	160 (9)
lin318	64	**18373 (171)**	35 (1)	19172 (311)	60 (1)	18613 (174)	154 (10)
rd400	81	**5880 (137)**	60 (3)	6242 (73)	108 (2)	5990 (84)	330 (23)
fl417	93	7936 (5)	69 (4)	7958 (17)	111 (3)	**7927 (3)**	408 (31)
gr431	87	980 (9)	64 (4)	993 (6)	117 (6)	**979 (2)**	396 (20)
pr439	96	55710 (655)	77 (5)	56832 (336)	126 (4)	**55638 (273)**	472 (24)
pcb442	96	**21544 (394)**	75 (4)	22462 (214)	131 (1)	21607 (195)	478 (29)
d493	102	**16655 (164)**	88 (7)	17202 (127)	164 (9)	17027 (76)	617 (43)
att532	110	39270 (591)	96 (1)	40525 (385)	182 (6)	**39077 (208)**	776 (52)
ali535	108	**1077 (4)**	94 (1)	1096 (5)	184 (8)	1086 (5)	745 (48)
u574	127	16518 (497)	122 (2)	16879 (160)	226 (10)	**16119 (136)**	1052 (85)
rat575	121	2412 (95)	114 (1)	2505 (33)	228 (6)	**2284 (24)**	995 (59)
p654	134	22462 (32)	140 (1)	22437 (20)	260 (8)	**22401 (3)**	1427 (79)
d657	137	**21532 (385)**	147 (1)	22402 (226)	294 (10)	22138 (191)	1442 (109)
gr666	139	**1577 (6)**	150 (1)	1614 (13)	278 (9)	1601 (15)	1504 (97)
u724	166	17660 (468)	208 (1)	18470 (212)	394 (13)	**17152 (136)**	2284 (162)
rat783	169	3398 (65)	236 (11)	3533 (39)	454 (17)	**3180 (28)**	2667 (224)

[1] The number of clusters
[2] Average weight (Standard Deviation)
[3] Average running time (Standard Deviation)

Table 7 Computational results for the proposed method, TS, and MOGA for TSPLIB instances with the grid clustering $\mu = 7$

Graph name	K[1]	Proposed LA-based		TS (Öncan et al. 2008)		MOGA (Contreras-Bolton et al. 2016)	
		Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]
gr229	34	**618** (**3**)	15 (1)	630 (3)	37 (1)	622 (4)	50 (2)
gil262	49	**845** (**7**)	22 (1)	913 (16)	45 (1)	881 (12)	90 (6)
pr264	42	**19999** (**88**)	18 (0)	20358 (187)	44 (1)	20102 (73)	76 (5)
pr299	47	**16415** (**136**)	22 (0)	16725 (368)	54 (0)	16785 (221)	99 (6)
lin318	49	**15476** (**141**)	24 (0)	15750 (187)	61 (1)	15484 (97)	108 (8)
rd400	64	**5033** (**182**)	42 (1)	5399 (72)	107 (1)	5184 (52)	260 (15)
fl417	61	**7481** (**24**)	40 (0)	7498 (22)	114 (4)	7487 (12)	262 (16)
gr431	64	878 (12)	43 (0)	890 (9)	123 (4)	**875** (**5**)	283 (19)
pr439	74	48727 (266)	51 (0)	49719 (420)	122 (2)	**48309** (**181**)	340 (18)
pcb442	64	**16236** (**190**)	45 (0)	17237 (340)	130 (1)	16736 (271)	320 (24)
d493	78	**14605** (**195**)	60 (0)	15017 (160)	155 (4)	14811 (146)	447 (30)
att532	80	**34013** (**445**)	66 (0)	35609 (477)	187 (6)	35005 (355)	537 (40)
ali535	83	**931** (**5**)	69 (1)	953 (6)	185 (8)	935 (3)	561 (38)
u574	92	**13334** (**110**)	82 (1)	14335 (174)	223 (9)	13653 (196)	743 (50)
rat575	100	2111 (106)	90 (2)	2254 (35)	215 (1)	**2056** (**34**)	821 (63)
p654	100	21699 (43)	97 (0)	21684 (20)	253 (7)	**21650** (**7**)	977 (67)
d657	96	**18204** (**587**)	95 (1)	18954 (265)	288 (9)	18622 (139)	946 (66)
gr666	96	1367 (25)	96 (1)	1367 (9)	283 (9)	**1341** (**5**)	933 (66)
u724	119	15103 (422)	132 (1)	15510 (156)	357 (9)	**14574** (**110**)	1393 (89)
rat783	121	2823 (121)	147 (6)	2922 (43)	403 (9)	**2633** (**82**)	1686 (139)

[1] The number of clusters
[2] Average weight (Standard Deviation)
[3] Average running time (Standard Deviation)

Table 8 Computational results for the proposed method, TS, and MOGA for TSPLIB instances with the grid clustering $\mu = 10$

Graph name	K[1]	Proposed LA-based		TS (Öncan et al. 2008)		MOGA (Contreras-Bolton et al. 2016)	
		Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]	Avg. weight[2]	CPU Time[3]
gr229	23	**514 (5)**	11 (0)	548 (10)	44 (1)	515 (4)	39 (2)
gil262	36	**679 (29)**	16 (0)	740 (18)	46 (1)	707 (13)	67 (4)
pr264	27	**16755 (124)**	13 (1)	16942 (202)	52 (1)	16868 (127)	53 (3)
pr299	35	12769 (201)	17 (0)	**12713 (392)**	58 (1)	12817 (235)	80 (6)
lin318	36	**11366 (716)**	18 (0)	12446 (353)	66 (1)	12017 (353)	91 (6)
rd400	49	**4180 (176)**	33 (0)	4478 (87)	109 (0)	4378 (65)	210 (14)
fl417	42	7022 (28)	31 (0)	7050 (27)	123 (3)	**7013 (8)**	191 (13)
gr431	52	782 (7)	36 (0)	793 (8)	125 (5)	**771 (4)**	227 (13)
pr439	48	**41565 (271)**	40 (1)	42381 (488)	126 (3)	41723 (219)	262 (21)
pcb442	48	**13044 (462)**	41 (1)	13973 (234)	133 (2)	13710 (225)	258 (25)
d493	52	**11628 (250)**	48 (1)	11971 (177)	168 (4)	11862 (102)	294 (24)
att532	57	**27952 (548)**	56 (1)	29493 (467)	189 (4)	28647 (273)	387 (36)
ali535	57	**739 (4)**	57 (1)	774 (13)	195 (8)	761 (7)	387 (30)
u574	64	**11034 (121)**	66 (1)	11656 (207)	213 (2)	11211 (251)	532 (29)
rat575	64	1599 (115)	63 (4)	1683 (53)	214 (1)	**1534 (32)**	486 (34)
p654	69	20828 (12)	74 (3)	20893 (52)	261 (7)	**20810 (12)**	731 (56)
d657	73	**15147 (318)**	77 (2)	16138 (302)	274 (2)	15678 (198)	683 (49)
gr666	70	**1134 (20)**	74 (2)	1172 (13)	278 (11)	1136 (4)	709 (41)
u724	80	11601 (478)	92 (1)	12060 (194)	333 (2)	**11344 (106)**	962 (80)
rat783	81	2152 (103)	10 (2)	2223 (46)	388 (1)	**1975 (29)**	1202 (84)

[1] The number of clusters
[2] Average weight (Standard Deviation)
[3] Average running time (Standard Deviation)

Table 9 The result of the Wilcoxon test based on the solution quality

Algorithms compared	p value	Significant[1]	Decision
$\mathbf{LA_c}$ versus $\mathbf{TS_c}$	0.000	Yes	Reject H_0
$\mathbf{LA_c}$ versus $\mathbf{MOGA_c}$	0.017	Yes	Reject H_0
$\mathbf{LA\text{-}_{=3}}$ versus $\mathbf{TS\text{-}_{=3}}$	0.000	Yes	Reject H_0
$\mathbf{LA\text{-}_{=3}}$ versus $\mathbf{MOGA\text{-}_{=3}}$	0.455	No	Fail to reject H_0
$\mathbf{LA\text{-}_{=5}}$ versus $\mathbf{TS\text{-}_{=5}}$	0.000	Yes	Reject H_0
$\mathbf{LA\text{-}_{=5}}$ versus $\mathbf{MOGA\text{-}_{=5}}$	0.654	No	Fail to reject H_0
$\mathbf{LA\text{-}_{=7}}$ versus $\mathbf{TS\text{-}_{=7}}$	0.000	Yes	Reject H_0
$\mathbf{LA\text{-}_{=7}}$ versus $\mathbf{MOGA\text{-}_{=7}}$	0.211	No	Fail to reject H_0
$\mathbf{LA\text{-}_{=10}}$ versus $\mathbf{TS\text{-}_{=10}}$	0.000	Yes	Reject H_0
$\mathbf{LA\text{-}_{=10}}$ versus $\mathbf{MOGA\text{-}_{=10}}$	0.0384	Yes	Reject H_0

[1] p value ≤ 0.05

Table 10 The result of the Wilcoxon test based on the running time

Algorithms compared	p value	Significant[1]
$\mathbf{LA_c}$ versus $\mathbf{TS_c}$	0.000	Yes
$\mathbf{LA_c}$ versus $\mathbf{MOGA_c}$	0.000	Yes
$\mathbf{LA\text{-}_{=3}}$ versus $\mathbf{TS\text{-}_{=3}}$	0.350	No
$\mathbf{LA\text{-}_{=3}}$ versus $\mathbf{MOGA\text{-}_{=3}}$	0.000	Yes
$\mathbf{LA\text{-}_{=5}}$ versus $\mathbf{TS\text{-}_{=5}}$	0.000	Yes
$\mathbf{LA\text{-}_{=5}}$ versus $\mathbf{MOGA\text{-}_{=5}}$	0.000	Yes
$\mathbf{LA\text{-}_{=7}}$ versus $\mathbf{TS\text{-}_{=7}}$	0.000	Yes
$\mathbf{LA\text{-}_{=7}}$ versus $\mathbf{MOGA\text{-}_{=7}}$	0.000	Yes
$\mathbf{LA\text{-}_{=10}}$ versus $\mathbf{TS\text{-}_{=10}}$	0.000	Yes
$\mathbf{LA\text{-}_{=10}}$ versus $\mathbf{MOGA\text{-}_{=10}}$	0.000	Yes

[1] p value ≤ 0.05

the p values were higher than the $\alpha = 0.05$ level. Therefore, the performance of the algorithms is similar in terms of the quality of the solution.

As the statistical results in Table 10 show, it can be concluded that the performance of the proposed algorithm is remarkable in terms of running time.

6 Conclusions

In this paper, an algorithm based on the learning automata is introduced to solve the GMST problem, which is denoted as LA-GMST. In the proposed algorithm, an LA is considered for each cluster. The number of LA actions is the same as the number of nodes in the corresponding cluster. All LAs select an action from their action-sets. The selected actions are used to determine one node from each cluster. Then, a MST is constructed on the selected nodes and its weight is taken into account for a reinforcement signal to LAs. This process continues until the desired GMST is obtained. The selection of a proper learning rate plays a crucial role in the efficiency of the algorithm. Therefore, its influence on the algorithm result is investigated. The results indicate that a high value for the learning rate conduces to a few repetitions to explore the best actions. It is worth mentioning that low value results in focusing on a few regions of the search space and will miss other potential regions. To evaluate the quality of the results returned by the proposed algorithm, it was compared to the two best existing algorithms, the TS-based and MOGA algorithm. The numerical experiments are performed on a large set of extended TSPLIB. The results indicate that the proposed algorithm performs better than the TS-based algorithm both in terms of both solution quality and running time. Although in terms of solution quality, the performance of the proposed and MOGA algorithms is highly competitive, the proposed algorithm dominates the compared algorithm regarding the running time. The difference between the running time of our algorithm and other algorithms becomes much more remarkable in a time-sensitive environment like cluster head selection in ad-hoc networks.

References

Akbari Torkestani J, Meybodi MR (2010a) An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata. Comput Netw 54(5):826–843

Akbari Torkestani J, Meybodi MR (2010b) Mobility-based multicast routing algorithm for wireless mobile Ad-hoc networks: a learning automata approach. Comput Commun 33(6):721–735

Akbari Torkestani J, Meybodi MR (2011) Learning automata-based algorithms for solving stochastic minimum spanning tree problem. Appl Soft Comput J 11(6):4064–4077

BoussaïD I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Inf Sci 237:82–117

Contreras-Bolton C, Gatica G, Rey C, Parada V (2016) A multi-operator genetic algorithm for the generalized minimum spanning tree problem. Expert Syst Appl 50:1–8

Daliri Khomami MM, Rezvanian A, Bagherpour N, Meybodi MR (2018) Minimum positive influence dominating set and its application in influence maximization: a learning automata approach. Appl Intell 48(3):570–593

Di C, Li S, Li F, Qi K (2019) A novel framework for learning automata: a statistical hypothesis testing approach. IEEE Access 7:27911–27922

Dror M, Haouari M (2000) Generalized steiner problems and other variants. J Comb Optim 4(4):415–436

Dua A, Sharma P, Ganju S, Jindal A, Aujla GS, Kumar N, Rodrigues JJ (2018) Rovan: a rough set-based scheme for cluster head selection in vehicular ad-hoc networks. In: 2018 IEEE global communications conference (GLOBECOM), IEEE, pp 206–212

Fahimi M, Ghasemi A (2017) A distributed learning automata scheme for spectrum management in self-organized cognitive radio network. IEEE Trans Mob Comput 16(6):1490–1501

Farman H, Jan B, Javed H, Ahmad N, Iqbal J, Arshad M, Ali S (2018) Multi-criteria based zone head selection in internet of things based wireless sensor networks. Future Gener Comput Syst 87:364–371

Ferreira CS, Ochi LS, Parada V, Uchoa E (2012) A GRASP-based approach to the generalized minimum spanning tree problem. Expert Syst Appl 39(3):3526–3536

García S, Molina D, Lozano M, Herrera F (2009) A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. J Heurist 15(6):617–644

Ghavipour M, Meybodi MR (2016) An adaptive fuzzy recommender system based on learning automata. Electron Commer Res Appl 20:105–115

Ghavipour M, Meybodi MR (2018) Trust propagation algorithm based on learning automata for inferring local trust in online social networks. Knowl Based Syst 143:317–326

Golden B, Raghavan S, Stanojević D (2005) Heuristic search for the generalized minimum spanning tree problem. INFORMS J Comput 17(3):290–304

Haouari M, Chaouachi JS (2006) Upper and lower bounding strategies for the generalized minimum spanning tree problem. Eur J Oper Res 171(2):632–647

Hasanzadeh-Mofrad M, Rezvanian A (2018) Learning automata clustering. J Comput Sci 24:379–388

Hu B, Leitner M, Raidl GR (2008) Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. J Heurist 14(5):473–499

Jiang W, Li B, Li S, Tang Y, Chen CLP (2016) A new prospective for learning automata: a machine learning approach. Neurocomputing 188:319–325

Krishna PV, Misra S, Nagaraju D, Saritha V, Obaidat MS (2016) Learning automata based decision making algorithm for task offloading in mobile cloud. In: IEEE CITS 2016—2016 international conference on computer, information and telecommunication systems

Kumar N, Misra S, Obaidat MS (2015) Collaborative learning automata-based routing for rescue operations in dense urban regions using vehicular sensor networks. IEEE Syst J 9(3):1081–1090

Misra S, Krishna PV, Kalaiselvan K, Saritha V, Obaidat MS (2014) Learning automata-based QoS framework for cloud IaaS. IEEE Trans Netw Serv Manag 11(1):15–24

Misra S, Venkata Krishna P, Saritha V, Agarwal H, Vasilakos AV, Obaidat MS (2017) Learning automata-based fault-tolerant system for dynamic autonomous unmanned vehicular networks. IEEE Syst J 11(4):1–10

Mollakhalili Meybodi MR, Meybodi MR (2014) Extended distributed learning automata: an automata-based framework for solving stochastic graph optimization problems. Appl Intell 41(3):923–940

Moradabadi B, Meybodi MR (2017) A novel time series link prediction method: learning automata approach. Physica A Stat Mech Its Appl 482:422–432

Moradabadi B, Meybodi MR (2018) Link prediction in weighted social networks using learning automata. Eng Appl Artif Intell 70(February):16–24

Mostafaei H (2015) Stochastic barrier coverage in wireless sensor networks based on distributed learning automata. Comput Commun 55:51–61

Mostafaei H, Meybodi MR (2013) Maximizing lifetime of target coverage in wireless sensor networks using learning automata. Wirel Pers Commun 71(2):1461–1477

Mostafaei H, Montieri A, Persico V, Pescapé A (2017) A sleep scheduling approach based on learning automata for WSN partial coverage. J Netw Comput Appl 80(December 2016):67–78

Mukherjee A, Keshary V, Pandya K, Dey N, Satapathy SC (2018) Flying ad hoc networks: a comprehensive survey. In: Information and decision sciences. Springer, pp 569–580

Myung Y-S, Lee C-H (1995) On the generalized minimum spanning tree problem. Networks 11(4):231–241

Nettleton DF (2013) Data mining of social networks represented as graphs. Comput Sci Rev 7:1–34

Öncan T, Cordeau J-F, Laporte G (2008) A tabu search heuristic for the generalized minimum spanning tree problem. Eur J Oper Res 191(2):306–319

Park J-H, Choi S-C, Hussen HR, Kim J (2017) Analysis of dynamic cluster head selection for mission-oriented flying ad hoc network. In: 2017 ninth international conference on ubiquitous and future networks (ICUFN). IEEE, pp 21–23

Pop PC, Kern W, Still G (2006) A new relaxation method for the generalized minimum spanning tree problem. Eur J Oper Res 170(3):900–908

Prabaharan G, Jayashri S (2019) Mobile cluster head selection using soft computing technique in wireless sensor network. Soft Comput 23(18):8525–8538

Ranjbari M, Akbari Torkestani J (2018) A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers. J Parallel Distrib Comput 113:55–62

Rao PS, Jana PK, Banka H (2017) A particle swarm optimization based energy efficient cluster head selection algorithm for wireless sensor networks. Wirel Netw 23(7):2005–2020

Reddy MPK, Babu MR (2019) A hybrid cluster head selection model for internet of things. Clust Comput 22(6):13095–13107

Ren J, Wu G, Su X, Cui G, Xia F, Obaidat MS (2018) Learning automata-based data aggregation tree construction framework for cyber-physical systems. IEEE Syst J 12(2):1467–1479

Rezvanian A, Meybodi MR (2015) Finding minimum vertex covering in stochastic graphs: a learning automata approach. Cybern Syst 46(8):698–727

Rezvanian A, Meybodi MR (2017) A new learning automata-based sampling algorithm for social networks. Int J Commun Syst 30(5):1–21

Sang Q, Lin Z, Acton ST (2016) Learning automata for image segmentation. Pattern Recognit Lett 74:46–52

Sengottuvelan P, Prasath N (2017) Bafsa: breeding artificial fish swarm algorithm for optimal cluster head selection in wireless sensor networks. Wirel Pers Commun 94(4):1979–1991

Shojafar M, Abolfazli S, Mostafaei H, Singhal M (2015) Improving channel assignment in multi-radio wireless mesh networks with learning automata. Wirel Pers Commun 82(1):61–80

Tang D, Liu X, Jiao Y, Yue Q (2011) A load balanced multiple cluster-heads routing protocol for wireless sensor networks. In: 2011 IEEE 13th international conference on communication technology. IEEE, pp 656–660

Tarabalka Y, Benediktsson JA, Chanussot J (2009) Spectral-spatial classification of hyperspectral imagery based on partitional clustering techniques. IEEE Trans Geosci Remote Sens 47(8):2973–2987