Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Finding minimum weight connected dominating set in stochastic graph based on learning automata

Javad Akbari Torkestani [a,*], Mohammad Reza Meybodi [b]

[a] *Young Researchers Club, Arak Branch, Islamic Azad University, Arak, Iran*
[b] *Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, Iran*

A R T I C L E   I N F O

Article history:
Received 14 November 2008
Received in revised form 15 January 2012
Accepted 27 February 2012
Available online xxxx

Keywords:
Dominating set
Minimum connected dominating set
Stochastic graph
Learning automata

A B S T R A C T

Finding the minimum weight connected dominating set (MCDS) in an arbitrary graph is an NP-hard problem and several heuristics and approximation methods have been proposed to solve it. Forwarding the messages along the virtual backbone induced by the connected dominating set (CDS) significantly reduces the routing overhead as well as the power consumption by reducing the routing nodes to the backbone nodes. This paper first defines the stochastic MCDS problem where the probability distribution function (PDF) of the random weight associated with the graph vertices is unknown. Then, it presents several learning automata-based algorithms (Algorithms 1–6) to solve the stochastic MCDS problem. Taking advantage of learning automata, the proposed algorithms significantly reduce the number of samples that must be taken from the graph to construct the MCDS. It is proved that by the proper choice of the learning rate, the probability of finding the MCDS is close enough to unity. The standard sampling method (SSM) is the baseline with which we compare the performance of the proposed algorithms. Experimental results show that Algorithm 6 significantly outperforms the SSM and the other proposed algorithms in terms of the sampling rate.

1. Introduction

The dominating set (DS) problems are a general class of the optimization problems that are widely used in wireless networking [4,6,7,12,14,16–18,20,22,27,30,37–40,42]. A DS of a given graph $G = (V,E)$ is a subset $S \subseteq V$ in such a way that every vertex $v \subseteq V$ is either in DS S or adjacent to a vertex of S. A vertex of S is said to dominate itself and all its adjacent vertices. Finding the dominating set of the network topology graph is a well-known approach for clustering the wireless networks [2,12,20,22,37,39]. The DS with the minimum cardinality is said to be a minimum DS (MDS). In [33], Pang et al. studied the dominating set in directed graphs and proposed an algorithm to compute a minimal dominating set on digraphs. It is an arc-insertion based method. A dominating set is also an independent dominating set, if no two vertices in the set are adjacent. A connected dominating set (CDS) S of a given graph G is a dominating set whose induced sub-graph, denoted $\langle S \rangle$, is connected, and a minimum size CDS is a CDS with the minimum cardinality. A minimum size CDS forms a virtual backbone in the network graph by which the routing overhead can be significantly reduced, where the number of hosts responsible for the route discovery and data transmission can be reduced to the number of vertices in the CDS [5,13,14,28]. The minimum size DS and minimum size CDS problems have been shown to be NP-Hard [16,30], and even for a unit disk graph, the problem of finding a minimum size CDS is still NP-Hard [30].

Since the characteristics of the wireless networks are stochastic, unpredictable and time-variable [2,8,15,24,31], the stochastic graph is a more appropriate structure to model the behavior of these network. The CDS problem in deterministic graphs

* Corresponding author.
E-mail address: j-akbari@iau-arak.ac.ir (J. Akbari Torkestani).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.02.057

has been widely studied and a host of solutions are available [6,11,22,29,35,36,38,39,41], but to the best of our knowledge, the idea of solving the MCDS problem in stochastic graphs is first discussed in this paper. Guha and Khuller [20] proposed two centralized greedy heuristics with bounded performance for CDS formation in vertex-weighted graphs. In the first algorithm, the CDS initially includes a single node. CDS grows from one node outward. The second algorithm first forms a WCDS. It then connects the WCDS nodes by the intermediate nodes. Butenko et al. [12] proposed a prune-based heuristic algorithm for construction of small CDSs. This algorithm initializes the CDS to the vertex set of the graph. Then, it prunes the CDS by removing some nodes. A given node must be retained, if the node elimination causes disconnection of the CDS. Otherwise, it can be removed. Wu et al. [39] also presented two pruning-based CDS algorithms by improving the distributed algorithm proposed by Wu and Li [38]. The algorithm proposed by Wu and Li [38] first constructs a CDS. Then, it prunes the CDS by removing some redundant nodes. In [35], Tsai et al. proposed an efficient algorithm for finding the minimum size CDS of a trapezoid graph with time complexity $O(n)$. They also designed an algorithm for solving the minimum weight CDS problem in $O(n \log \log n)$ time.

Li et al. [26] presented a polynomial-time $(3H(n-1)-1)$-approximation algorithm for solving the minimum CDS, where H denotes the harmonic function. They claim that the proposed algorithm can be used in asymmetric multi-hop wireless networks. The approximation ratio of the proposed algorithm is a factor of 3 from the best possible approximation algorithm. Zou et al. [44] proposed a $(5+\varepsilon)$-approximation method for solving the MCDS problem in a vertex-weighted graph. The proposed method is composed of two main phases. The first phase presents a $(4+\varepsilon)$-approximation dynamic programming algorithm for solving the minimum DS problem and the second phase presents a $(1+\varepsilon)$ -approximation algorithm to solve the Steiner tree problem to connect the DS nodes.

Alzoubi et al. [4–6] proposed two weighted heuristics for constructing CDS in a wireless ad hoc network. Both algorithms initially employ the distributed leader election algorithm given in [4] to construct a rooted spanning tree [19] from the original network topology. Then, an iterative labeling strategy is used to classify the nodes of the tree to the dominators or dominatees based on their ranks. The first heuristic uses an ID-based approach for ranking the nodes and the second one uses a level-based approach. Alzoubi et al. [3] proposed another distributed algorithm for approximating the minimum CDS in a wireless ad hoc network. The proposed algorithm first constructs a maximal independent set (MIS) and then connects the MIS nodes by a Steiner tree. Han [22] proposed a zone-based distributed CDS algorithm for backbone formation in wireless ad hoc networks. The proposed algorithm uses the node degree as the dominator selection criterion. Wang et al. [37] proposed a distributed weighted algorithm for constructing the CDS with the minimum cost in a vertex-weighted graph. Yin et al. [43] presented an asynchronous distributed single-phase algorithm called DSP-CDS for constructing a CDS in ad hoc networks. In this algorithm, each node uses only the information of its one-hop neighbors to make a local decision on joining the dominating set. Therefore, it is well adapted to the highly dynamic networks.

The previous CDS formation algorithms assume that the graph is deterministic. That is, the graph characteristics (e.g., the weight associated with the graph vertices) are fixed and do not vary over time. This assumption cannot hold true in many cases such as wireless networks where the network parameters are affected by many factors like environmental influences, conflicts, collisions, and so forth. Therefore, this paper first introduces the stochastic version of the MCDS problem. It then presents several learning automata-based heuristic algorithms for solving the stochastic MCDS problem. In this paper, it is assumed that the PDF of the random weight of the vertices is unknown. This assumption makes the stochastic MCDS problem much harder to solve. Each algorithm aims at reducing the number of samples that must be taken from the graph to find the MCDS. The preliminary version of the first proposed algorithm (Algorithm 1) appeared in [1]. The results given in [1] show the superiority of the learning automata-based MCDS formation algorithm over the standard sampling method in terms of the sampling rate. Ref. [1] studies the impact of the learning rate on the performance of the proposed algorithm from an experimental point of view. It shows that the convergence rate of the algorithm to the optimal solution increases as the learning rate decreases. From a theoretical point of view, this paper aims to prove that by the proper choice of the parameters of Algorithm 1, this algorithm would be able to construct the MCDS with a probability close enough to unity. Several simulation experiments are performed to investigate the performance of the proposed algorithms. The obtained results show the superiority of the proposed algorithms over the standard sampling method in terms of the sampling rate.

The rest of the paper is organized as follows. The stochastic graph and theory of learning automata are described in the next section. In Section 3, the proposed learning automata-based algorithms are presented. Convergence of Algorithm 1 to the optimal solution is shown in Section 4. The relationship between the learning rate and convergence rate of Algorithm 1 is also discussed in this section. The performance of the proposed algorithms is evaluated through the simulation experiments in Section 5. Section 6 concludes the paper.

2. Stochastic graphs, and learning automata theory

To provide the sufficient background for understanding the CDS formation algorithms presented in this paper, the stochastic graph and learning automata theory are briefly reviewed in this section.

2.1. Stochastic graphs

A vertex-weighted graph can be modeled by a triple $G = \langle \mathbf{V}, \mathbf{E}, W \rangle$, where $V = \{v_1, v_2, \ldots, v_n\}$ denotes the set of graph vertices, $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V} = \{e_1, e_2, \ldots, e_m\}$ denotes the edge set, and $W = \{w_1, w_2, \ldots, w_n\}$ denotes the set of vertex weights. w_i is the weight associated with vertex v_i for $i \in \{1,2,\ldots,n\}$. A vertex-weighted graph G is stochastic and hereafter in this paper is referred to as stochastic graph, if the weight of vertex v_i is a random variable with PDF w_i. Let $G = \langle \mathbf{V}, \mathbf{E}, W \rangle$ be a stochastic graph and $\Omega = \{\omega_1, \omega_2, \ldots, \omega_r\}$ denotes the set of all its CDSs. Let $\overline{W}(v_i)$ be the expected weight of vertex v_i and $\overline{W}(\omega_i) = \sum_{v_i \in \omega_i} \overline{W}(v_i)$ denotes the expected weight of CDS ω_i. The CDS $\omega^* \in \Omega$ of stochastic graph G is minimum, if and only if $\overline{W}(\omega^*) = \min_{\forall \omega_i \in \Omega} \overline{W}(\omega_i)$. That is, The MCDS of a given stochastic graph G is defined as the stochastic CDS with the minimum expected weight.

2.2. Learning automata

A learning automaton [32] is an adaptive decision-making system that can improve its performance by learning how to choose the optimal action from a set of allowed actions through repeated interactions with the random environment. Automaton randomly chooses its action based on a PDF defined over the action-set. At each iteration, the selected 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 [32].

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ denotes the finite set of inputs, $\beta \equiv \{\beta_1, \beta_2, \ldots, \beta_m\}$ is 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. Depending on the nature of the reinforcement signal $\underline{\beta}$, environments can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as$P$-model environments. Another class of the environment allows a finite number of the values in the interval $[0,1]$ can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval $[a,b]$. The random environment is said to be a non-stationary environment, if the penalty probabilities vary over time, and is called stationary otherwise. The relationship between the learning automaton and the random environment is shown in Fig. 1.

Learning automata can be classified into two main families [45,46]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $\langle \underline{\beta}\,\underline{\alpha}, L \rangle$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and L is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha_i(k) \in \underline{\alpha}$ and $\underline{p(k)}$ denote the action selected by learning automaton and the probability vector defined over the action set at instant k, respectively. Let a and b denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. Let r be the number of actions that can be taken by the learning automaton. At each instant k, the action probability vector $\underline{p(k)}$ 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.

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

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

If $a = b$, the recurrence Eqs. (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 ($L_{R-\varepsilon P}$), 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. The following describes some convergence results of the learning automata.

Definition 2.1. The average penalty probability $M(n)$, received by a given automaton is defined as

$$M(n) = E[\beta(n)|\zeta_n] = \int_{\alpha \in \underline{\alpha}} \zeta_n(\alpha) f(\alpha)$$

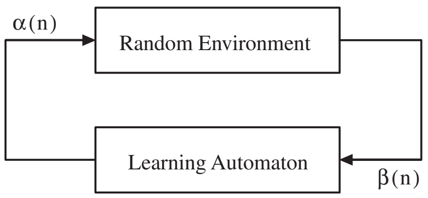


Fig. 1. The relationship between the learning automaton and its random environment.

where ζ: $\underline{\alpha} \to [0,1]$ specifies the probability of choosing each action $\alpha \in \underline{\alpha}$, and $\zeta_n(\alpha)$ is called the action probability.

If no priori information is available about f, there is no basis for selection of action. So, all the actions are selected with the same probabilities. This automaton is called *pure chance automaton* and its average penalty is equal to

$$M_0 = E[f(\alpha)]$$

Definition 2.2. A learning automaton operating in a P-, Q-, or S-model environment is said to be *expedient* if

$$\lim_{n\to\infty} E[M(n)] < M_0$$

Expediency means that when automaton updates its action probability function, its average penalty probability decreases. Expediency can also be defined as a closeness of $E[M(n)]$ to $f_l = \min_{\alpha} f(\alpha)$. It is desirable to take an action by which the average penalty can be minimized. In such case, the learning automaton is called *optimal*.

Definition 2.3. A learning automaton operating in a P-, Q-, or S-model environment is said to be *absolutely expedient* if

$$E\lfloor M(n+1)|\underline{p}(n)\rfloor < M(n)$$

for all n and all $p_i(n)$.

Absolute expediency implies that $M(n)$ is a super martingale and $E[M(n)]$ is strictly decreasing for all n in all stationary environments. If $M(n) \leqslant M_0$, absolute expediency implies expediency.

Definition 2.4. A learning automaton operating in a P-, Q-, or S-model environment is said to be *optimal* if

$$\lim_{n\to\infty} E[M(n)] = f_l$$

Optimality implies that asymptotically the action for which penalty function attains its minimum value is chosen with probability one. While optimality appears a very desirable property, certain conditions in a given situation may preclude its environment. In such cases, a suboptimal performance is desirable. Such property is called ε-optimality and is defined in the following definition

Definition 2.5. A learning automaton operating in a P-, Q-, or S-model environment is said to be ε-*optimal* if

$$\lim_{n\to\infty} E[M(n)] < f_l + \varepsilon$$

can be obtained for any $\varepsilon > 0$ by a proper choice of the parameters of the learning automaton.

ε-optimality implies that the performance of the learning automaton can be made as close to the optimal as desired.

2.2.1. Distributed learning automata

A learning automaton is a simple agent for doing simple things. The full potential of learning automata is realized when they are interconnected to work together. A Distributed learning automata (DLA) [9] is a network of learning automata cooperating to solve a particular problem. Formally, a DLA can be defined by a quadruple $\langle A, E, L, A_0 \rangle$, where $A = \{A_1, \ldots, A_n\}$ denotes the set of learning automata, $E \subset A \times A$ is the set of the edges (edge $e_{(i,j)}$ corresponds to action α_j of the automaton A_i), L denotes the set of learning schemes with which the learning automata update their action probability vectors, and A_0 is the root automaton from which the activation of the learning automata begins. An example DLA is shown in Fig. 2.

A DLA operates as follows: At first, the root automaton randomly chooses one of its outgoing edges (i.e., actions) according to its action probability vector. The root automaton activates the automaton at the other end of the selected edge. The activated automaton also randomly selects an action activating another automaton. This process continues until a leaf automaton (an automaton that interacts to the environment) is activated. The selected actions along the path induced by the activated automata are applied to the random environment. The environment evaluates the applied actions and emits a reinforcement signal to the DLA. The activated learning automata update their action probability vectors on the basis of

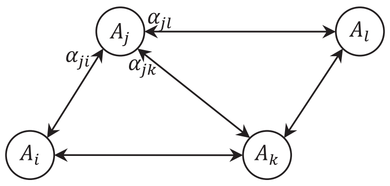


Fig. 2. Distributed learning automata.

the reinforcement signal. Different paths connecting the root automaton to the leaf automata are selected and updated according to the environment feedback until the choice probability of a path converges to one.

2.2.2. 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 [34] 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, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. $A = \{A_1, A_2, \ldots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $q(k) = \{q_1(k), q_2(k), \ldots, q_m(k)\}$ defined over the possible subsets of the actions, where $q_i(k) = prob[A(k) = A_i | A_i \in A, 1 \leqslant i \leqslant 2^n - 1]$. $\hat{p}_i(k) = prob[\alpha(k) = \alpha_i | A(k), \alpha_i \in A(k)]$ is the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and also $\alpha_i \in A(k)$. The scaled probability $\hat{p}_i(k)$ is defined as

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

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 $p_i(k) = prob[\alpha(k) = \alpha_i]$.

The procedure of choosing an action and updating the action probability vector in a variable action-set learning automaton can be described as follows. Let $A(k)$ be the action subset selected at instant k. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in 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 [34].

3. The proposed CDS formation algorithms

In this section, six learning automata-based algorithms are proposed for solving the minimum CDS problem in stochastic graphs as described in Subsection 2.1. For all algorithms, the vertex-weighted stochastic graph $G\langle V, E, W\rangle$ is served as the input of algorithm and the output is the CDS of having the minimum expected weight. In all algorithms, the PDF of the random weight associated with each vertex is a priori unknown and must be estimated by a statistical method. The response of the environment is the same for all algorithms. At first, a network of the learning automata is formed by assigning a learning automaton A_i to each vertex $v_i \in V$. That is, A_i and v_i are two notations for the same concept and so might be used instead of each other in the text. The resultant network of automata can be described by a triple $\langle A, \alpha, W\rangle$, where $A = \{A_1, A_2, \ldots, A_n\}$ represents the set of the learning automata corresponding to the graph vertices, $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ denotes the action-set (where $\alpha_i = \{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^{r_i}\}$ defines the set of actions that can be taken by learning automaton A_i, for each $\alpha_i \in \alpha$), and $W = \{w_1, \ldots, w_n\}$ denotes the set of weights associated with set A (or corresponding to vertex-set V) such that $w_i = W(A_i)$ (for every $A_i \in A$) is the random weight associated with automaton A_i. In the first proposed algorithm which is called Algorithm 1, the action-set of each learning automaton A_i which is referred to as α_i, contains only two actions α_i^0 and α_i^1. Learning automaton A_i assigns the role of a dominatee to vertex v_i by selection of action α_i^0 and the role of a dominator by choosing action α_i^1. Fig. 3 shows the high level pseudo code of Algorithm 1. This algorithm consists of a number of stages. At each stage, the graph vertices assuming the role of a dominator form the connected dominating set. This algorithm iteratively constructs a number of CDSs and updates the action probability vectors until it finds a near optimal solution to the minimum weight CDS problem. Stage k of the proposed algorithm is briefly described in the following steps:

```
Step 1. CDS formation
   Repeat
 – For all learning automata do in parallel
    – Each automaton $A_i$ chooses one of its two actions according to its action probability vector
    – If the chosen action declares vertex $v_i$ as a dominator, then
       – Vertex $v_i$ is added to the set of dominator vertices being selected in this stage
       – Vertex $v_i$ and all its neighbors are added to the set of dominatee vertices (if they have not been already added)
       – The random weight associated with selected action is added to the weight of the selected dominating set

   Until the cardinality of dominatee set is equal to the number of vertices of $G$ & the selected dominators form a
   connected sub-graph
Step 2. Comparing the weight of the constructed CDS with a dynamic threshold
 – Let CDS $\omega_i$ be the selected CDS at stage $k$.
 – The average weight of the constructed CDS $\omega_i$ until stage $k$ is computed as
```

(continued on next page)

Algorithm 1 The first algorithm proposed for finding the MCDS in a stochastic graph

```
01:Input Stochastic Vertex-Weighted Graph $G =< V, E, W >$, Thresholds $K$, $P$
02:Output The minimum weight connected dominating set ;
03:Assumptions
04:   Assign an automaton $A_i$ to each vertex $v_i$
05:   Let $\alpha_i = \{\alpha_i^0, \alpha_i^1\}$ denotes the action-set of automaton $A_i$
06:Begin
07:   Let dynamic threshold $T_k$ be the average weight of all CDSs constructed until stage $k$ and initially is set
to $n$
08:   Let $k$ denotes the stage number and is initially set to 0
09:   Repeat
10:        Let $d^k$ denotes the set of dominatee vertices constructed at stage $k$
11:        Let $\omega^k$ be the CDS constructed at stage $k$
12:        Let $W^k$ be the weight of CDS constructed at stage $k$
13:        Let $\overline{W}(\omega^k)$ be the average weight of all samples taken from $\omega^k$ and initially is set to 0
14:        While ($|d^k| < n$ or the dominators do not form a connected graph) Do
15:             For all the vertices of the stochastic graph do in parallel
16:                  Learning automaton $A_i$ randomly chooses one of its two actions
17:                  If (The chosen action is $\alpha_i^1$) Then
18:                       $\omega^k \leftarrow \omega^k + v_i$
19:                       Vertex $v_i$ and all its neighbors are added to $d^k$
20:                       $W^k \leftarrow W^k + w_i$
21:                  End If
22:             End For
23:        End While
24:        Compute $\overline{W}(\omega^k)$ by Equation (4)
25:        If ($\overline{W}(\omega^k) < T_{k-1}$) Then
26:             The chosen actions by all the learning automata are rewarded
27:        Else
28:             The chosen actions by all the learning automata are penalized
29:        End If
30:        Update $T_k$ by Equation (6)
31:        $k \leftarrow k + 1$
32:   Until ($PCDS > P$ or $k > K$)
33:End Algorithm
```

Fig. 3. The pseudo code of Algorithm 1.

$$\overline{W}(\omega_i^k) = \frac{1}{n_i^k}\sum_{j=1}^{n_i^k} W(\omega_i(j)) \tag{4}$$

where n_i^k denotes the number of times CDS ω_i is constructed until stage k, and $W(\omega_i(j))$ denotes the weight of the jth sampled CDS ω_i, which is defined as

$$W(\omega_i(j)) = \sum_{v_s \in \omega_i} W(v_s(j)) \tag{5}$$

where $W(v_s(j))$ denotes the weight of the vertex v_s in the jth sampled ω_i.

– The average weight computed for the constructed CDS ω_i is compared to the dynamic threshold T_k. At stage $k > 1$, the value of the dynamic threshold T_k is calculated as

$$T_k = \frac{1}{r}\sum_{i=1}^{r} \overline{W}(\omega_i^k) \tag{6}$$

where r denotes the number of all CDSs constructed until stage k.

Step 3. Updating the action probability vectors

– Depending on the result of Step 2, all learning automata (*only the activated automata in Algorithms 4, 5, and 6*) reward their chosen actions if the average weight of the constructed CDS is less than or equal to the dynamic threshold T_{k-1} and penalize them otherwise. Each learning automaton updates its action probability vector by using a L_{R-I} reinforcement scheme.

Step 4. Stop condition

– The CDS construction process and updating the action probabilities continue until the product of the probability of choosing the vertices of the CDS called *PCDS* is greater than a certain threshold or the number of constructed CDS exceeds a pre-specified threshold. The CDS that is formed just before the algorithm stops is the CDS with the minimum expected weight among all CDSs of the stochastic graph.

The first algorithm has two disadvantages that may increase the running time of algorithm due to redundant samples. The first weakness point is that the set of dominators that is selected at each stage is not necessarily a dominating set. The second one is that the sub-graph induced by the constructed DS may be disconnected. The second algorithm which is proposed for solving the MCDS problem tries to resolve the first disadvantage of Algorithm 1. This algorithm, which is called Algorithm 2, differs from Algorithm 1 in action-set and the CDS formation method (Step 1). In Algorithm 2, the action-set of each learning automaton A_i is defined as follows: Action-set of learning automaton A_i includes an action for learning automaton A_i and an action for each learning automaton A_j, where vertex v_j is adjacent to vertex v_i and $j \in \{1,\ldots,n\}$. That is, the action-set of learning automaton A_i is defined as $\alpha_i = \{A_j \mid \forall j; j = i \text{ or } (v_i, v_j) \in E\}$. Such an action-set means that each learning automaton either declares itself as a dominator or selects one of its neighbors as its dominator. Generally, at each stage$k$, choosing action $A_j \in \alpha_i$ declares vertex v_j (corresponding to the chosen action A_j) as a dominator and adds it to the CDS which is being formed in that stage. Fig. 4 shows the CDS formation step of Algorithm 3 at stage k. The remaining steps (*steps 2, 3, and 4*) of Algorithm 2 are the same as those described in Algorithm 1.

Another solution that is proposed for the first disadvantage of Algorithm 1 is based on distributed learning automata described in Subsection 2.2.1. In this algorithm, which is called *Algorithm 3*, each learning automaton can be in one of two states *active* and *passive*. All learning automata are initially set to the passive state. Algorithm 3 activates the learning automata that are selected at each stage. Each activated automaton is added to the set of dominators. Algorithm 3 forms the action-set of each learning automaton as follows: Learning automaton A_i reserves an action for all learning automata excluding itself. That is, the action-set of automaton A_i is defined as $\alpha_i = \{A_j \mid \forall j \in V, j \neq i\}$. In Algorithm 3, the first dominator is randomly selected by a separate learning automaton whose action-set includes all vertices of vertex-set V. The selected automaton is activated and chooses one of its actions as a dominator. The selected vertex and its neighbors are added to the dominatee set. The activation process continues until the cardinality of the dominatee set is equal to the cardinality of vertex-set V. Depending on the response received from the random environment, the proposed algorithm rewards or penalizes the activated automata. The CDS formation step of Algorithm 3 at each stage k is shown in Fig. 5.

In Algorithm 3, the number of actions that is taken by each learning automaton is assumed to be fixed and does not vary with time. The fixed action-set learning automaton used in this algorithm significantly increases the sampling rate of algorithm. To reduce the sampling rate of Algorithm 3, we propose *Algorithm 4* in which the number of actions available for each learning automaton changes with time. In this algorithm, each passive learning automaton A_i changes its number of actions (or scales its action probability vector) by disabling the actions corresponding to all dominatees as described in Subsection 2.2.2 on variable action-set learning automata.

The first four proposed algorithms do not guarantee to form a connected DS at each stage. This significantly increases the number of samples must be taken in the first step of these algorithms. To solve the addressed problem, we propose a DLA-based algorithm (called *Algorithm 5*) by which the formation of a CDS is guaranteed at each stage. The action-set of each learning automaton A_i includes learning automaton A_j (for all $j \in \{1,\ldots,n\}$) assigned to vertex v_j, where vertex v_j is adjacent to vertex v_i. At each stage, Algorithm 5 starts by randomly choosing one of the graph vertices as dominator. The learning automaton corresponding to the selected vertex is activated and chooses one of its actions based on its action probability vector. The action corresponding to the activated learning automaton is disabled in the action-set of all learning automata. The (learning automaton) activation process continues until the formation of a virtual backbone (or constructing a CDS) or no more actions can be taken by the currently active automaton. In the former case, the current stage terminates by finding a solution to the minimum weight CDS problem. In the latter case, the algorithm traces the path induced by the activated automata back for finding a learning automaton with available actions. An available action is the action that has not been already disabled. The action-set of such an automaton must be pruned by removing the last chosen action. Current stage is resumed by this automaton as described earlier. In this algorithm, each learning automaton may activate one or more neighbors at each stage. That is, more than one action can be chosen by each learning automaton. The CDS formation step of Algorithm 5 has been described in more detail in Fig. 6.

It can be seen that Algorithm 5 may select a number of dominators by which no more vertex can be added to the dominatee set. Obviously, this increases the cardinality of the CDS and the rate of unnecessary samples considerably. To relieve this problem, in Algorithm 5, all learning automata disable the actions by which no more vertices can be covered. This algorithm which we call it Algorithm 6 significantly reduces the size of CDS as compared to Algorithm 5.

Algorithm 2. Step 1 The CDS formation step of Algorithm 2

```
01:Repeat
02:  For all learning automata do in parallel
03:      Each automaton $A_i$ chooses one of its actions according to its action probability vector
04:      The vertex corresponding to the selected action is added to CDS
05:      The random weight of the selected vertex is added to the weight of CDS
06:  End for
07:Until the sub-graph induced by the selected dominators is connected
```

Fig. 4. The pseudo code of the CDS formation step of Algorithm 2.

Algorithm 3. Step 1 The CDS formation step of Algorithm 3

```
01:Repeat
02:   $v_i$ is randomly selected as the first dominator and added to the CDS
03:   While the cardinality of the constructed dominatee set is less than $n$ Do
04:        $v_i$ and its neighbors are added to the dominatee set
05:        Random weight associated with $v_i$ is added to the weight of CDS
06:        $A_i$ randomly chooses one of its actions
07:        The vertex corresponding to the selected action is denoted as $v_j$
08:        $v_j$ is added to the CDS
09:        $v_i \leftarrow v_j$
10:   End while
11:Until the sub-graph induced by the selected dominators is connected
```

Fig. 5. The pseudo code of the CDS formation step of Algorithm 3.

4. Convergence results

This section proves two main results of the paper. First, the convergence of Algorithm 1 to the optimal solution by the proper choice of learning rate a when a L_{R-I} learning algorithm is used at each node. Second, determination of a learning rate under which the probability of constructing (or converging to) the CDS with the minimum expected weight is larger than $1-\varepsilon$. The second result concerns the relationship between the convergence error parameter ε (i.e., the error parameter involved in the standard sampling method) and the learning rate of Algorithm 1.

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

$$Prob\left[\lim_{k\to\infty} q_i(k) = 1\right] \geqslant 1-\varepsilon \qquad (7)$$

Proof. The method that is used to prove Theorem 1 is similar to the method given in [9,32] to analyze the behavior of the learning automaton operating in a non-stationary environment. The steps of the convergence proof are briefly outlined as follows. At first, it is proved that, the penalty probabilities of the CDSs converge to the constant values of the final penalty probabilities, for large enough values of k (Lemma 1). Then, it is shown that, the probability of choosing the CDS with the minimum expected weight, is a sub-Martingale process for large values of k, and so the changes in the probability of constructing the minimum CDS is always non-negative (Lemmas 2 and 3). Finally, the convergence of the Algorithm 1 to the minimum CDS is proved by using Martingale convergence theorems. Therefore, the following lemmas need to be proved before stating the proof of the theorem. □

Lemma 1. *If the CDS ω_i is penalized with probability $c_i(k)$ at stage k and $Lim_{k\to\infty} c_i(k) = c_i^*$, then for each $\varepsilon \in (0,1)$ and $k > K(\varepsilon)$ we have*

$$-prob\left[|c_i^* - c_i(k)| > 0\right] < \varepsilon$$

where $K(\varepsilon)$ denotes the minimum number of stages of Algorithm 1 to achieve error rate ε.

Algorithm 5. Step 1 The CDS formation step of Algorithm 5

```
01:$v_i$ is randomly selected as the first dominator and added to the CDS
02:While the cardinality of the constructed dominatee set is less than $n$ Do
03:   $v_i$ and its neighbors are added to the dominatee set
04:   All learning automata disable the action corresponding to the selected vertex $v_i$
05:   Random weight associated with $v_i$ is added to the weight of CDS
06:   If $|\alpha_i| \neq 0$ then
07:        $A_i$ randomly chooses one of its actions
08:   Else
09:        Find an automaton with available actions among the activated automata and denote it $A_i$
10:        $A_i$ randomly chooses one of its actions
11:   The vertex corresponding to the selected action is denoted as $v_j$
12:   $v_j$ is added to the CDS
13:   $v_i \leftarrow v_j$
14:End while
```

Fig. 6. The pseudo code of the CDS formation step of Algorithm 5.

Proof. Before stating the proof of Lemma 1, the convergence of $c_i(k)$ to c_i^* (as k converges to infinity) is discussed. From Algorithm 1, we find that the probability of penalizing CDS ω_i at stage k is defined as $c_i(k) = prod[\overline{W}\omega_i > T_k]$. In other words, each activated automaton penalizes its selected action, if the average weight of the constructed CDS is greater than the dynamic threshold T_k. In the beginning of algorithm (for small values of k), the values of dynamic threshold T_k and average weight $\overline{W}\omega_i$ are very far from their true values. This causes some near to optimal CDS is penalized, while the CDSs with higher weights are rewarded. As algorithm proceeds (i.e., as k becomes larger), the number of samples taken from the graph increases, and so T_k and $\overline{W}\omega_i$ converge to their true values. This means that the algorithm knows the behavior of the random environment more and more as k grows. In other words, k as becomes larger, the probability of penalizing the optimal or near optimal CDS decreases and that of the others increases. Therefore, for large enough values of k, the probability of penalizing CDS ω_i (i.e., $ci(k)$) converges to its true value c_i^*. Let r denotes the number of all constructed CDSs, and c_i^* denotes the final value of probability $c_i(k)$ when k is large enough. Using weak law of large numbers, we conclude that

$$Lim_{k\to\infty} prob\left[\left|c_i^* - c_i(k)\right| > \varepsilon\right] \to 0 \tag{8}$$

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

Lemma 2. *Let $c_j(k) = prob[\overline{W}\omega_j(k+1) > T_k]$ and $d_j = 1 - c_j$ be the probability of penalizing and rewarding the CDS ω_j (for all $j = 1,2,\ldots,r$), respectively. If $\underline{q}(k)$ evolves according to Algorithm 1, then the conditional expectation of $q_i(k)$ is defined as*

$$E[q_i(k+1)|q(k)] = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_i(k) + d_j(k)\left(\prod_{v_m \notin \omega_i}\left(1 - p_m^1(k)\right)\prod_{v_m \in \omega_i} p_m^1(k)\right)\right] \tag{9}$$

where

$$p_m^1(k+1) = \begin{cases} p_m^1(k) + a(1 - p_m^1(k)); & v_m \in \omega_j \\ p_m^1(k)\cdot(1-a); & v_m \notin \omega_j \end{cases}$$

where $p_m^1(k)$ is the probability that vertex v_m declares itself as a dominator vertex, at stage k, and r denotes the number of all CDSs constructed until stage k. $v_m \in \omega_i$, if it declares itself as a dominator vertex (or chooses action α_m^1) and $v_m \notin \omega_i$ otherwise.

Proof. Algorithm 1 uses a L_{R-I} reinforcement scheme to update the probability vector at each stage$k$. Therefore, the construction probability of the CDS ω_i (i.e., $q_i(k)$) remains unchanged with probability $c_j(k)$ (for all $j = 1,2,\ldots,r$) when the constructed CDS ω_j is penalized by the random environment. On the other hand, when the constructed CDS ω_j is rewarded, the probability of choosing the vertices of the CDS ω_i, which are in the constructed CDS ω_j, increases by a given learning rate as that of the other vertices decreases.

Let

$$q_i(k) = \prod_{v_m \notin \omega_i} p_m^0(k) \prod_{v_m \in \omega_i} p_m^1(k) \tag{10}$$

be the probability of constructing minimum CDS ω_i at stage k, where p_m^0 and p_m^1 denote the probability with which vertex v_m declares itself as a dominatee vertex and as a dominator, respectively. The conditional expectation of $q_1(k+1)$, assuming $\underline{q}(k)$ evolves according to Algorithm 1, is defined as

$$E[q_1(k+1)|q(k)] = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_1(k) + d_j(k)\left(\prod_{v_m \notin \omega_1} p_m^0(k) \prod_{v_m \in \omega_1} p_m^1(k)\right)\right]$$

where

$$p_m^0(k+1) = \begin{cases} p_m^0(k) + a\left(1 - p_m^0(k)\right); & v_m \notin \omega_j \\ p_m^0(k)\cdot(1-a); & v_m \in \omega_j \end{cases}$$

$$p_m^1(k+1) = \begin{cases} p_m^1(k) + a\left(1 - p_m^1(k)\right); & v_m \in \omega_j \\ p_m^1(k)\cdot(1-a); & v_m \notin \omega_j \end{cases}$$

Since each learning automaton has only two possible actions, the probability with which vertex v_m declares itself as a dominatee vertex (i.e., p_m^0) is defined as $1 - p_m^1$. Hence, q_i at stage k can be defined as $\prod_{v_m \notin \omega_i}(1 - p_1^m(k))\prod_{v_m \in \omega_i} p_1^m(k)$, and so we have

$$E[q_i(k+1)|q(k)] = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_i(k) + d_j(k)\left(\prod_{v_m \notin \omega_i}\left(1 - p_1^m(k)\right)\prod_{v_m \in \omega_i} p_1^m(k)\right)\right]$$

where

$$p_m^1(k+1) = \begin{cases} p_m^1(k) + a(1 - p_m^1(k)); & v_m \in \omega_j \\ p_m^1(k) \cdot (1-a); & v_m \notin \omega_j \end{cases}$$

and hence the proof of the lemma. □

Lemma 3. *Let $q_i(k)$ denotes the probability of formation of the minimum CDS ω_i, at stage k. The increment in the conditional expectation of $q_i(k)$ is always non-negative, assuming $\underline{q}(k)$ is updated according to Algorithm 1. That is, $\Delta q_i(k) > 0$.*

Proof. Define

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

From Lemma 2, we have

$$\Delta q_i(k) = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_i(k) + d_j(k)\left(\prod_{v_m \notin \omega_i}(1 - p_m^1(k))\prod_{v_m \in \omega_i} p_m^1(k)\right)\right] - q_i(k)$$

where

$$p_m^1(k+1) = \begin{cases} p_m^1(k) + a(1 - p_m^1(k)); & v_m \in \omega_j \\ p_m^1(k) \cdot (1-a); & v_m \notin \omega_j \end{cases}$$

Let $\prod_{n\in\{0,1\}}^{\omega_i} \pi_m^n(k) = \prod_{v_m \notin \omega_i}(1 - p_m^1(k))\prod_{v_m\in\omega_i} p_m^1(k)$, for notational convenience. Therefore, we have

$$\Delta q_i(k) = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_i(k) + d_j(k)\prod_{n\in\{0,1\}}^{\omega_i} \pi_m^n(k)\right] - q_i(k) \tag{11}$$

where

$$\pi_m^n(k+1) = \begin{cases} p_m^n(k) + a(1 - p_m^n(k)); & (v_m \notin \omega_j, n=0) \text{ or } (v_m \in \omega_j, n=1) \\ p_m^n(k) \cdot (1-a); & (v_m \notin \omega_j, n=1) \text{ or } (v_m \in \omega_j, n=0) \end{cases}$$

where $\pi_m^n(k)$ is the probability with which vertex v_m declares itself as a dominator (when $n = 1$) and as a dominatee vertex otherwise. The probability of constructing, rewarding, and penalizing a CDS is defined as the product of the probability of declaring the vertices of the CDS as dominators or dominatees. Therefore, we have

$$\Delta q_i(k) = \sum_{j=1}^{r}\prod_{n\in\{0,1\}}^{\omega_j} p_m^n(k)\left[\prod_{n\in\{0,1\}}^{\omega_j} c_m^n(k)\prod_{n\in\{0,1\}}^{\omega_i} p_m^n(k) + \prod_{n\in\{0,1\}}^{\omega_j} d_m^n(k)\prod_{n\in\{0,1\}}^{\omega_i}\pi_m^n(k)\right] - \prod_{n\in\{0,1\}}^{\omega_i} p_m^n(k)\cdot$$

where $\pi_m^n(k)$ is defined as given in Eq. (11), $c_m^n(k)$ is the probability of penalizing the action that is chosen by vertex v_m at stage k, and $d_m^n(k) = 1 - c_m^n(k)$.

$$\Delta q_i(k) = \prod_{n\in\{0,1\}}^{\omega_i} E\left[p_m^n(k+1)|p_m(k)\right] - \prod_{n\in\{0,1\}}^{\omega_i} p_m^n(k)$$

The equality above can be rewritten as

$$\Delta q_i(k) \geqslant \prod_{n\in\{0,1\}}^{\omega_i}\left(E\left[p_m^n(k+1)|p_m(k)\right] - p_m^n(k)\right) = \prod_{n\in\{0,1\}}^{\omega_i} \Delta p_m^n(k) \tag{12}$$

and

$$\Delta \underset{n\in\{0,1\}}{p_m^n}(k) = a\cdot p_m^n(k)\sum_{s\neq n} p_m^s(k)\cdot\left(c_m^s(k) - c_m^n(k)\right)$$

$q_i(k) \in (0,1)$ for all $\underline{q} \in S_r^0$, where $S_r = \left\{\underline{q}(k) : 0 \leqslant q_i(k) \leqslant 1; \sum_{i=1}^{r} q_i(k) = 1\right\}$ and S_r^0 denotes the interior of S_r. Hence, $p_m^n(k) \in (0,1)$ for all m, n. Since action α_m^n is the action with the minimum penalty probability which can be selected by automaton A_m, it is shown that $c_m^{s^*} - c_m^{n^*} > 0$, for all $s \neq n$, where $c_m^{s^*}$ is the final value to which the penalty probability $c_m^{s^*}$

is converged. It follows from Lemma 1 that for large values of k, $c_m^s(k) - c_m^n(k) > 0$. Therefore, we conclude that for large values of k, the right hand side of the equation above consists of the non-negative quantities, and so we have

$$\prod_{\substack{\omega_i \\ n\in\{0,1\}}} a \cdot p_m^n(k) \sum_{s\neq n} p_m^s(k) \cdot \left(c_m^s(k) - c_m^n(k)\right) \geqslant 0$$

and from Eq. (12), we have

$$\Delta q_i(k) \geqslant \prod_{\substack{\omega_i \\ n\in\{0,1\}}} a \cdot p_m^n(k) \sum_{s\neq n} p_m^s(k) \cdot \left(c_m^s(k) - c_m^n(k)\right)$$

which completes the proof of this lemma. □

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

Proof. This corollary has been shown in [32]. □

Let $\Gamma_i(q)$ be the probability with which the Algorithm 1 converges to the unit vector e_i with initial probability vector $\underline{q}$ and defined as

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

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

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

where $E[.]$ represents the mathematical expectation. It has been shown in [32] that operator U is linear and as the expectation of a non-negative function remains non-negative, operator U preserves the non-negative functions. In other word, $U\psi(q) \geqslant 0$ for all $q \in S_r$, if $\psi(q) \geqslant 0$. If the operator U is repeatedly applied $n$times (for all $n > 1$), we have

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

A function $\psi(q)$ is called super-regular (sub-regular) if and only if $\psi(q) \geqslant U\psi(q)(\psi(q) \leqslant U\psi(q))$, for all $q \in S_r$. It has also been shown in [32] that $\Gamma_i(q)$ is the only continuous solution of $U\Gamma_i(q) = \Gamma_i(q)$, with the following boundary conditions.

$$\begin{aligned} &\Gamma_i(e_i) = 1 \\ &\Gamma_i(e_j) = 0; \quad j \neq i \end{aligned} \tag{14}$$

Define

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

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

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

$$\psi_i(q) \geqslant \Gamma_i(q)$$

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

$$\psi_i(q) \leqslant \Gamma_i(q) \tag{15}$$

for all $q \in S_r$.

Proof. Theorem 2 has been proved in [32]. □

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

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

is super-regular. We now determine the conditions under which $\theta_i\ [x,q]$ is super-regular. From the definition of operator Ugiven in Eqs. (13) and (11) we have

$$U\theta_i(x,q) = E\left[e^{-\frac{xq_i(k+1)}{a}} | q(k) = q\right] = \left[\sum_{j=1}^{r} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{v_m \notin \omega_j, n=0 \\ or \\ v_m \in \omega_j, n=1}} \left(p_m^n + a(1-p_m^n)\right)\right]} + \sum_{j=1}^{r} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{v_m \notin \omega_j, n=1 \\ or \\ v_m \in \omega_j, n=0}} \left(p_m^n(1-a)\right)\right]}\right]$$

$$= \left[q_i d_i^* e^{-\frac{x}{a}(q_i + a(1-q_i))} + \sum_{j \neq i} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{v_m \notin \omega_j, n=0 \\ or \\ v_m \in \omega_j, n=1}} \left(p_m^n + a\left(1-p_m^n\right)\right)\right]} + \sum_{j \neq i} q_j d_j^* e^{-\frac{x}{a}\left[\prod_{\substack{v_m \notin \omega_j, n=1 \\ or \\ v_m \in \omega_j, n=0}} \left(p_m^n(1-a)\right)\right]}\right]$$

where d_j^* denotes the final value to which the reward probability $d_j(k)$ is converged, for large values of k, and $e^{-\frac{x}{a}(q_i+a(1-q_i))}$ is the expectation of $\theta_i\,(x,q)$, when the minimum CDS ω_i is rewarded by the environment and it is calculated as the product of the rewarded probabilities of the dominator or dominatee vertices (the vertices which declare themselves as dominator and dominatees) which constructs CDS ω_i.

$$U\theta_i(x,q) = \left[q_i d_i^* e^{-\frac{x}{a}(q_i + a(1-q_i))} + \sum_{j \neq i} q_j d_j^* e^{-\frac{x}{a}\left(q_i(1-a) \cdot \prod_{\substack{v_m \notin \omega_j, n=0 \\ or \\ v_m \in \omega_j, n=1}} \frac{\left(p_m^n + a\left(1-p_m^n\right)\right)}{\left(p_m^n(1-a)\right)}\right)}\right]$$

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

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

$$\rho_i^j = \begin{cases} \prod_{\substack{v_m \notin \omega_j, n=0 \\ or \\ v_m \in \omega_j, n=1}} \frac{(p_m^n + a(1-p_m^n))}{(p_m^n(1-a))}; & i \neq j \\ 1; \quad i = j \text{ or } (\omega_i \cap \omega_j) = \phi \end{cases}$$

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

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

$$e^{-\frac{xq_i\rho_i^j}{a}} \sum_{j=i} q_j d_j^* e^{-x(1-q_i)\rho_i^j} + e^{-\frac{xq_i\rho_i^j}{a}} \sum_{j \neq i} q_j d_j^* e^{xq_i\rho_i^j} \leqslant e^{-\frac{xq_i}{a}}$$

and

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

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

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

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

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

and

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

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

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

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

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

$$G_i(x,q) \geqslant 0 \tag{17}$$

for all $q \in S_r$. From Eq. (16), it follows that $\theta_i\ (x,q)$ is super-regular if we have

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

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

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

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

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

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

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

For further simplification, let employ logarithms. Let

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

It has also been shown in [9,25] that

$$-\int_0^x H'(u)du \leqslant \Delta(q,x) \leqslant -\int_{-x}^0 H'(u)du$$
$$H(u) = \tfrac{dH(u)}{du}, \quad H(u) = \ln V(u)$$

We have

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

and

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

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

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

is a sub-regular function satisfying the boundary conditions (14) and from Theorem 2 and inequality (15), we conclude that

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

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

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

Thus we conclude that the probability with which the Algorithm 1 converges to (or constructs) the CDS with the minimum expected weight is equal to 1 as $k \to \infty$, and this completes the proof of the Theorem 1. □

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

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

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

Proof. It has been proved in [25,32] that there always exists a $x > 0$ under which the Eq. (19) is satisfied, if $d_j/d_i < 1$, for all $j \neq i$. Hence, it is concluded that

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

where q_i is the initial probability of the optimal CDS ω_i. It is assumed in Theorem 1 that the probability with which the Algorithm 1 converges to the CDS with the minimum expected weight is $(1 - \varepsilon)$, for each $0 < a < a^*$, where $a^*(\varepsilon) \in (0,1)$. So we conclude that,

$$\frac{1 - e^{-xq_i}}{1 - e^{-x}} = 1 - \varepsilon$$

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

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

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

5. Numerical results

To study the performance of the proposed algorithms, two different groups of simulation experiments are conducted. The first group of experiments aims to investigate the efficiency of the proposed algorithms on two stochastic graphs borrowed from [24], and the second group evaluates the scalability of the proposed algorithms on the connected dense stochastic graphs. Standard sampling method quoted in Appendix A is the baseline with which the results of the proposed algorithms are compared. In this paper, sampling rate and convergence rate are the metrics of interest. The reinforcement scheme used for updating the action probability vectors is L_{R-I}. Each algorithm terminates when the probability of the constructed CDS ($PCDS$) is 0.9 or greater or the number of constructed CDSs exceeds a pre-defined threshold. The first group of the simulation experiments tests the proposed algorithms on Graphs 1 and 2 for learning rate a ranging from 0.004 to 0.10. The obtained results are summarized in Tables 2 and 4 for the different values of learning rate a. Experimental results given in Tables 2 and 4 are compared with those of the standard sampling method given in Tables 1 and 3. The aim of using the standard sampling method is to provide a standard baseline for the sampling rate such that the average weight of the samples taken from the optimal CDS falls in the interval $(\mu - \delta, \mu + \delta)$ with probability $1 - \varepsilon$, where ε is the error parameter and δ is a small enough positive number. Graph1 which is shown in Fig. 7 has 8 vertices, 12 edges and its minimal CDS is $\delta_1^* = \{v_3, v_6\}$, and Graph 2 which is shown in Fig. 8 has 10 vertices, 21 edges and its minimal CDS is $\delta_2^* = \{v_3, v_7\}$. The probability distribution of the vertex weight of these two stochastic graphs are given in tables shown in Figs. 8 and 9.

Simulation results summarized in Tables 2 and 4 are averaged over 1000 independent runs. Each table includes the information about the total number of samples taken from the graph vertices (TS), the total number of samples taken from the

Table 1
The number of samples taken from Graph 1 in SSM.

Vertex	Confidence level for CDS										
	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.925	0.95	0.975	0.99
v_1	317	286	259	308	243	282	269	346	310	448	369
v_2	521	518	541	466	474	468	501	517	608	454	450
v_3	353	337	373	333	381	354	353	421	415	492	465
v_4	406	395	345	377	418	309	448	309	436	391	491
v_5	590	697	642	685	630	808	753	631	610	729	699
v_6	340	265	304	273	314	241	315	356	336	371	376
v_7	311	320	277	312	315	328	377	330	354	392	339
v_8	333	369	377	395	377	471	388	423	376	387	439
Total	3171	3187	3118	3149	3152	3261	3404	3333	3445	3664	3628

Table 2
Average number of samples taken from Graph 1 and CDS, and percentage of the converged runs.

Learning rate	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5			Algorithm 6		
	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC
0.004	4685.4	1800.3	100.0	27705.0	23477.0	100.0	3889.3	1812.6	100.0	2423.9	1808.0	100.0	4817.1	2773.2	100.0	2041.6	667.9	100.0
0.005	3753.8	1443.7	100.0	16421.0	12480.1	100.0	3130.7	1454.7	100.0	1944.4	1450.2	100.0	4017.9	2376.0	100.0	1691.7	570.2	100.0
0.006	3141.8	1199.2	100.0	14467.0	11409.3	100.0	2598.5	1211.7	100.0	1622.8	1214.2	100.0	2412.8	1085.8	100.0	1515.7	574.2	100.0
0.007	2697.2	1022.5	100.0	11242.9	8485.8	100.0	2237.4	1040.2	100.0	1392.8	1036.9	100.0	2065.4	954.7	100.0	1493.6	684.3	100.0
0.008	2364.7	889.5	100.0	10559.1	8046.5	100.0	1961.4	910.9	100.0	1215.3	904.0	100.0	1685.2	681.6	98.0	1271.8	529.6	95.0
0.009	2106.3	788.6	100.0	7293.5	5106.2	100.0	1745.9	810.2	100.0	1085.2	808.3	100.0	1528.1	597.0	94.0	1045.3	416.1	92.4
0.01	1890.3	711.2	100.0	4487.7	2577.0	100.0	1575.2	731.8	100.0	977.7	727.9	100.0	1396.4	589.1	95.2	934.6	364.0	89.6
0.02	957.3	343.6	100.0	3337.3	2279.4	93.2	786.7	359.9	100.0	486.5	360.3	100.0	616.4	216.6	90.0	450.7	156.5	84.5
0.03	642.0	225.7	99.7	1616.9	934.1	77.0	527.4	240.8	100.0	325.4	239.6	100.0	377.3	116.9	88.1	276.6	93.1	83.8
0.04	484.1	166.8	98.3	954.7	469.3	64.6	401.0	181.7	100.0	243.6	178.9	100.0	257.0	68.0	79.9	217.5	70.5	75.8
0.05	393.2	135.0	93.7	720.0	363.0	42.6	317.0	140.5	100.0	193.7	142.1	100.0	201.9	52.9	75.1	157.2	46.2	81.1
0.06	333.9	106.4	90.6	519.1	234.9	35.2	266.1	115.1	99.5	160.4	115.7	100.0	161.2	41.3	67.9	130.3	40.7	73.5
0.07	284.5	95.1	88.6	409.6	181.4	21.9	234.1	102.4	99.0	140.2	100.0	99.5	122.2	29.8	61.9	109.5	32.6	70.8
0.08	255.0	84.1	77.8	326.0	135.1	17.5	204.8	87.1	98.7	122.9	86.9	99.6	103.4	25.7	59.4	93.3	28.0	69.7
0.09	222.7	79.7	76.5	273.0	112.6	10.1	183.1	77.2	98.3	107.7	76.2	98.8	90.6	21.9	54.5	77.5	22.9	62.2
0.10	207.8	71.1	70.6	231.8	92.4	11.6	166.8	68.6	97.9	98.2	68.4	97.6	79.9	18.4	50.5	70.3	22.6	61.9

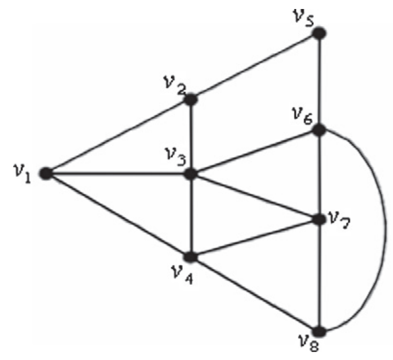
Table 3
The number of samples taken from Graph 2 in SSM.

Vertex	Confidence level for CDS										
	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.925	0.95	0.975	0.99
v_1	317	286	259	308	243	282	269	346	310	448	369
v_2	521	518	541	466	474	468	501	517	608	454	450
v_3	353	337	373	333	381	354	353	421	415	492	465
v_4	406	395	345	377	418	309	448	309	436	391	491
v_5	590	697	642	685	630	808	753	631	610	729	699
v_6	340	265	304	273	314	241	315	356	336	371	376
v_7	311	320	277	312	315	328	377	330	354	392	339
v_8	333	369	377	395	377	471	388	423	376	387	439
v_9	242	250	300	343	312	371	339	294	328	363	433
v_{10}	386	425	465	480	485	446	454	464	437	439	502
Total	3802	3867	3887	3977	3953	4084	4201	4096	4214	4469	4568

vertices of the CDS with the minimum weight (CDS), and the percentage of the converged runs (the percentage of runs converged to the minimum weight CDS) (PC). The results of the standard sampling method for Graphs 1 and 2 are given in Tables 1 and 3, respectively.

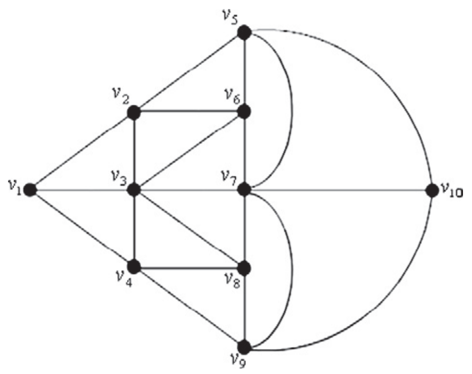
From the results of the simulation experiments, the following points can be made. The results showed that for all algorithms the total number of converged runs (convergence rate) increases as the learning rate decreases. For example, in Algorithm 1 for Graph 1, the number of converged runs is 89%, when $\alpha = 0.07$, whereas it increases to 100% when $\alpha = 0.007$ (see Table 2).

Experiments show that Algorithm 2 has a lower convergence rate compared to Algorithm 1. This might be due to the fact that the action-set of each learning automaton of Algorithm 2 is significantly larger than that of Algorithm 1, specifically for



Vertex	Weight	Probability
v_1	{2, 8, 12}	{0.9, 0.08, 0.02}
v_2	{10, 24, 35}	{0.85, 0.12, 0.03}
v_3	{6, 18, 24}	{0.88, 0.1, 0.02}
v_4	{12, 22, 30}	{0.85, 0.11, 0.04}
v_5	{17, 35, 50}	{0.75, 0.2, 0.05}
v_6	{3, 7, 10}	{0.68, 0.25, 0.07}
v_7	{4, 19, 15}	{0.75, 0.14, 0.11}
v_8	{5, 10, 12}	{0.65, 0.23, 0.12}

Fig. 7. Stochastic Graph 1 and its probability distribution function.



Vertex	Weight	Probability
v_1	{2, 8, 12}	{0.9, 0.08, 0.02}
v_2	{10, 24, 35}	{0.85, 0.12, 0.03}
v_3	{6, 18, 24}	{0.88, 0.1, 0.02}
v_4	{12, 22, 30}	{0.85, 0.11, 0.04}
v_5	{17, 35, 50}	{0.75, 0.2, 0.05}
v_6	{3, 7, 10}	{0.68, 0.25, 0.07}
v_7	{4, 19, 15}	{0.75, 0.14, 0.11}
v_8	{5, 10, 12}	{0.65, 0.23, 0.12}
v_9	{10, 19, 24}	{0.80, 0.14, 0.06}
v_{10}	{18, 27, 36}	{0.94, 0.05, 0.01}

Fig. 8. Stochastic Graph 2 and its probability distribution function.

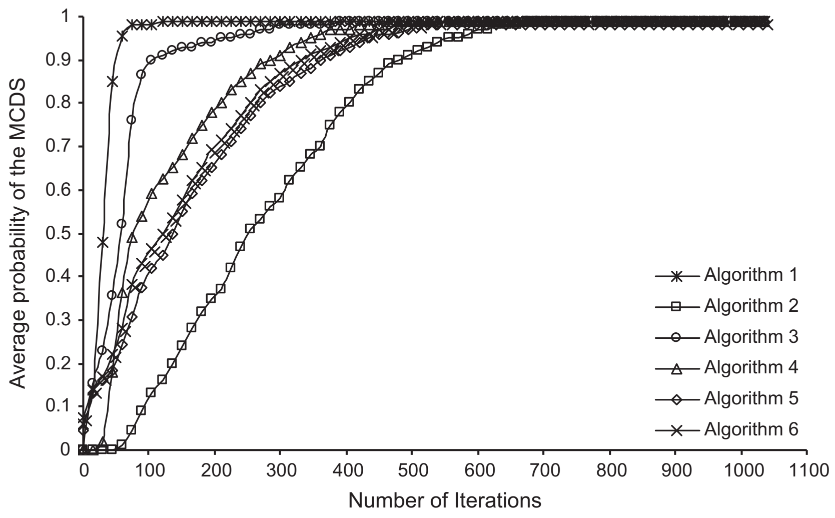


Fig. 9. The choice probability of the MCDS for Graph 1.

the dense graphs. As stated in Section 3, each learning automaton of Algorithm 2 has many choices (the number of actions available for each automaton equals to the number of its neighbors plus one), while in Algorithm 1 each learning automaton has at most two possible actions. A learning automaton with a larger action-set has a longer delay to converge to the optimal action. Therefore, the running time of Algorithm 2 is significantly longer than that of Algorithm 1.

Comparing the results given in Tables 2 and 4, it is observed that the total number of samples taken by Algorithm 3 is less than that of Algorithm 1 for the same convergence rate. The reason for this reduction in sampling rate can be due to the fact that Algorithm 3 guarantees to form a DS at each stage, while Algorithm 1 may fail in constructing the DS for several stages.

Table 4
The average number of samples taken from Graph 2 and MCDS, and the percentage of the converged runs for Graph 2.

Learning rate	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5			Algorithm 6		
	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC	TS	CDS	PC
0.004	5028.3	1435.5	100.0	46884.0	40789.0	100.0	4308.6	1777.4	100.0	2433.8	1772.4	100.0	5773.5	3381.0	100.0	4118.0	2125.0	100.0
0.005	4028.7	1144.1	100.0	35876.0	30060.0	100.0	3445.9	1424.5	100.0	1960.2	1425.2	100.0	4243.5	2350.8	100.0	3626.6	1921.5	100.0
0.006	3367.6	945.0	100.0	29630.0	25822.0	100.0	2884.0	1194.5	100.0	1620.7	1174.1	100.0	3275.4	1686.0	100.0	2869.1	1549.7	100.0
0.007	2892.2	798.3	100.0	21897.0	17836.0	100.0	2471.2	1019.1	100.0	1388.9	1009.8	100.0	2920.4	1582.1	100.0	2637.6	1549.8	100.0
0.008	2535.2	694.5	100.0	18052.0	13919.0	100.0	2174.0	889.5	100.0	1227.9	892.9	100.0	2403.3	1224.0	98.0	2180.1	1214.3	100.0
0.009	2256.8	618.7	100.0	12293.0	7891.0	100.0	1927.6	792.0	100.0	1092.7	793.9	100.0	2254.4	1176.3	94.0	1840.6	953.6	96.0
0.01	2147.2	914.7	100.0	5443.9	2591.3	100.0	1737.7	710.6	100.0	982.6	712.4	100.0	1791.9	826.2	93.7	1580.5	782.6	94.0
0.02	1085.4	444.8	100.0	4355.7	2917.9	77.2	877.6	354.8	100.0	496.7	357.3	100.0	694.0	219.1	87.7	578.2	191.1	90.0
0.03	728.6	288.6	97.9	2313.5	1414.1	46.3	588.5	234.4	100.0	327.9	235.1	100.0	418.2	108.2	80.8	380.4	120.7	82.0
0.04	551.0	213.5	95.9	1417.5	796.7	26.9	441.3	174.8	100.0	248.4	176.3	99.9	288.6	62.2	73.2	249.5	66.1	74.2
0.05	449.0	171.0	90.8	930.4	477.3	15.9	353.9	137.3	99.4	199.2	141.1	99.7	226.2	46.3	69.5	192.1	47.5	71.8
0.06	376.1	147.7	82.9	655.3	304.8	10.0	300.7	116.0	99.6	164.2	114.7	99.8	172.5	33.4	59.7	148.6	35.7	64.3
0.07	325.6	132.6	74.1	502.8	212.0	7.6	259.5	97.8	98.9	141.7	97.9	99.3	145.5	28.1	57.0	123.5	28.7	60.0
0.08	286.7	118.6	65.5	395.7	156.9	5.4	232.6	85.3	98.3	123.7	84.9	98.5	122.7	23.4	49.4	104.7	24.2	52.5
0.09	254.3	106.3	62.2	337.8	140.1	2.5	205.4	76.4	98.1	110.5	75.0	96.6	112.3	21.1	44.7	88.5	21.1	51.6
0.10	231.6	108.5	50.0	277.0	105.5	2.4	186.0	65.7	96.9	101.0	67.5	95.5	92.4	17.9	40.5	78.0	18.1	44.8

The obtained results show that Algorithm 4 greatly improves the convergence rate compared to Algorithm 3. The reason for the higher convergence rate is that Algorithm 4 employs the variable action-set learning automata to construct the CDSs. Variable action-set learning automata used in Algorithm 4 keep the actions that form the minimum CDS with a higher probability and removes the other actions. Therefore, Algorithm 4 increases the rate of convergence to the optimal CDS. The results show that Algorithm 4 also considerably outperforms Algorithms 1, 2, 3, and 5 in terms of the sampling rate.

Comparing the results of Algorithms 5 and 6 given in Tables 2 and 4, we find that Algorithm 6 has a higher convergence rate and lower sampling rate compared to Algorithm 5. This might be due to the fact that Algorithm 6 avoids sampling a large number of extra dominators (which are sampled in Algorithm 5) that do not cover more vertices. This property also reduces the sampling rate of Algorithm 6. Another reason for the outperformance of Algorithm 6 is the size of action-set. The learning automata used in Algorithm 6 have much smaller action-sets in comparison with Algorithm 5. Smaller action-set increases the convergence speed of the learning automaton and reduces the running time of algorithm. Therefore, Algorithm 6 significantly reduces the rate of unnecessary samples and improves the convergence rate too.

According to the results given in Tables 2, 4 and 6, Algorithm 5 has a higher convergence rate than both Algorithms 1 and 3. The number of samples needs to be taken by this algorithm to construct the minimum CDS is also much less than that of Algorithms 1 and 3.

Comparing the results of Tables 1 and 3 with Tables 2 and 4, we find that the sampling rate of the proposed algorithms (when the algorithms converge to the minimum CDS with probability $1-\varepsilon$) is considerably less than that of the standard sampling method. This is the main result of this paper that is proved in Section 4 and discussed in more detail at the end of this section.

The continuous-time convergence behavior of the proposed algorithms is shown in Figs. 9 and 10. This behavior is represented as the changes of the choice probability of the minimum CDS over time. From Figs. 9 and 10, a rapid progress can be seen in the beginning of each curve. Thereafter, all curves are flattened. This is a normal behavior for many iterative algorithms trying to improve the initial solution. The run time can be divided into two equally long sections, the first half and the second half. As Figs. 9 and 10 show, the choice probability of the minimum CDS increases much faster in the beginning. From the convergence behavior of the algorithms, it is concluded that it might not be worthwhile to allow very long runs. Because the long runs do not lead to a better solution in many cases. Every point along the continuous-time curve shows the average number of iterations required to find the minimum CDS with a certain probability. The average choice probability of the minimum CDS over the converged runs (out of 100 runs) for both graphs are shown in Figs. 9 and 10. Comparing the convergence behavior of the proposed algorithms, we observe that Algorithm 1 performs best and Algorithm 2 performs worst among the algorithms presented in this paper.

To evaluate the scalability of the proposed algorithms, in the second group of the experiments, the proposed algorithms are tested on the dense stochastic graphs generated based on the method described in [23] to create the weighted geometric graphs. The weight of each vertex v_i (for all $1 \leqslant i \leqslant n$) is assumed to be an exponential random variable with parameter $\lambda = i/n$, where n denotes the number of graph vertices. In experiments, the parameter of the underlying probability distribution of the vertex weight is assumed to be unknown. In a geometric graph, the vertices are randomly and uniformly distributed in a square area. An edge connects two given vertices, if they are close enough (if the distance between to vertices is less than r). In this experiment, the simulation square area is of size 100×100 unit and two vertices are connected together if their Euclidean distance is at most 30. The sampling rate of different algorithms is summarized in Table 5, where the graph size ranges from 20 to 120 nodes and convergence rate is 98 percent. All numerical results given in Table 5 are averaged over 100 runs on 100 different geometric stochastic graphs. Table 5 includes the information about the total number of samples taken from the graph (TS) and the total number of samples taken from the minimum CDS (CDS) for each of the proposed

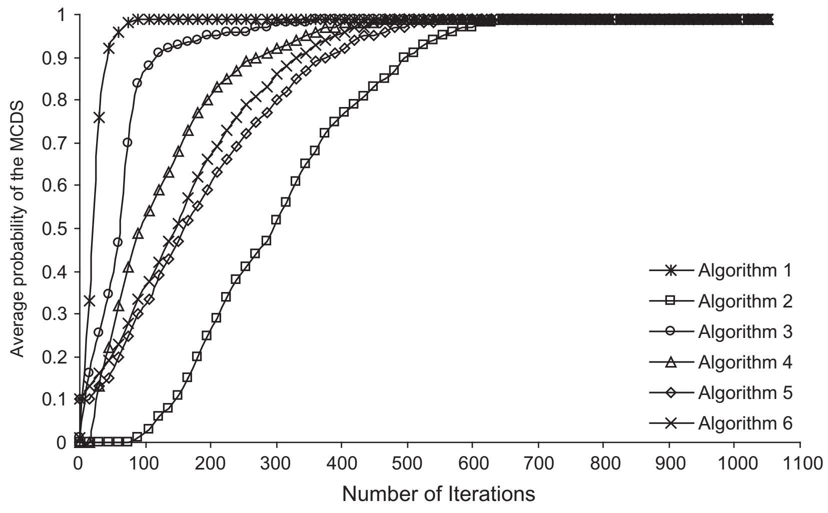


Fig. 10. The choice probability of the MCDS for Graph 2.

Table 5
The sampling rate of the proposed algorithms against the SSM.

Number of vertices	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 5		Algorithm 6		Standard sampling method
	TS	CDS	TS	CDS	TS	CDS	TS	CDS	TS	CDS	TS	CDS	
20	4891.30	829.00	6123.50	4210.00	2449.90	994.20	1488.60	761.60	2413.00	438.00	1056.00	408.00	7663.00
30	6084.80	1106.10	9367.50	4750.50	2556.20	744.70	2210.80	846.60	2546.00	260.00	1610.00	244.00	13186.00
40	6711.70	1003.70	13965.00	9410.00	3103.30	861.80	2255.33	756.67	2395.00	358.00	1769.00	156.00	18949.00
50	7820.00	968.60	13671.00	9005.50	3975.80	790.00	2443.00	871.00	3040.00	411.00	2139.00	103.50	22134.00
60	8251.00	991.50	16878.40	11231.00	3452.80	847.40	2810.00	749.33	3292.00	739.00	2490.00	932.00	26894.00
70	9464.60	1025.00	19076.00	14259.00	3579.00	685.00	2451.33	882.67	3599.00	820.00	2642.00	676.00	30335.00
80	9897.40	1111.10	25570.50	15590.20	3752.00	680.00	3453.00	804.00	3650.00	745.00	2951.00	460.00	33561.00
90	10100.60	916.00	36526.10	24560.50	4776.00	760.00	3495.00	1261.00	3874.00	745.00	3362.50	241.00	39505.00
100	11010.67	1110.33	37863.50	25670.90	5035.00	1707.00	4177.00	2022.00	4046.00	912.00	3556.00	581.00	43686.00
110	11334.17	908.50	42750.00	30410.00	5562.00	2568.00	4230.00	1750.00	4289.30	834.50	3845.50	612.50	49083.00
120	12041.83	890.33	45709.50	31902.50	5640.00	1290.50	4580.50	1520.00	4419.50	820.40	3770.20	600.20	50969.00

algorithms. It also includes the total number of samples taken by the standard sampling method when the confidence level is 98 percent. The numerical results given in Table 5 show the outperformance of Algorithm 6 over the other proposed algorithms in terms of the sampling rate. The obtained results also show that Algorithm 2 has the highest sampling rate. Comparing the average sampling rate of the proposed algorithms with that of the standard sampling method, we observe that all the proposed algorithms significantly outperform the standard sampling method, especially for the dense graphs.

To investigate the statistical significance of the results of the proposed algorithms, we used t-test for comparing the sampling rate of each pair of the proposed algorithm and the standard sampling method. This test is performed on the obtained results for geometric graphs where the number of vertices is 50 and the confidence level is set to 0.95. In this test, we assume that the difference between each pair of algorithm is statistically significant, if the difference significance is smaller than 0.05. The test results are summarized in Table 6. The first column (Algorithm x) of this table includes the list of the proposed algorithms together with the standard sampling method. The second column shows the mean sampling rate ± standard deviation for each algorithm x. This column shows that standard sampling method has the highest sampling rate and Algorithm 6 has the lowest rate. The third column contains the test results. In this column, the results are given for each pair of algorithm (x,y). Symbol "$\surd$" (or "$\boldsymbol{x}$") appeared in the column labeled as "Performance" indicates that algorithm x performs better (or worse) than algorithm y in terms of sampling rate. This conclusion is drawn on the basis of the mean sampling rate $(x - y)$ and difference significance. For instance, mean sampling rate $(x - y)$ of -578.00 and difference significance of 1.20E$-$12 show that Algorithm 4 outperforms Algorithm 5. Based on the test results given in the third column (labeled as "Test results"), the ranking of each algorithm is computed and summarized in the last column of Table 6. As expected, Algorithm 6 has the best result in terms of sampling rate and is ranked at the highest position and standard sampling method has the worst results [21].

The last experiment conducted in this paper aims at verifying the correctness of Theorem 3 empirically. This experiment compares the first proposed algorithm (*Algorithm 1*) with the standard sampling method in terms of sampling rate. Let q_i denotes the initial probability of constructing the minimum CDS ω_i, and $(1 - \varepsilon)$ denotes the probability with which

Table 6
Statistical significance of the sampling rate.

Algorithm x	Mean sampling rate x ($\pm SD$)	Test results				Ranking
		Algorithm y	Mean sampling rate $(x - y)$	Difference significance	Performance	
Algorithm 1	7705.00 (±178.65)	Algorithm 2	−5843.00	4.69E−23	√	5
		Algorithm 3	3836.00	2.07E−19	x	
		Algorithm 4	5214.00	1.69E−17	x	
		Algorithm 5	4636.00	8.22E−18	x	
		Algorithm 6	5609.00	1.53E−16	x	
		SSM	−14325.00	1.25E−26	√	
Algorithm 2	13548.00 (±199.54)	Algorithm 3	9679.00	8.93E−24	x	6
		Algorithm 4	11057.00	4.88E−20	x	
		Algorithm 5	10479.00	9.18E−21	x	
		Algorithm 6	11452.00	7.15E−19	x	
		SSM	−8482.00	1.14E−23	√	
Algorithm 3	3869.00 (±96.66)	Algorithm 4	1378.00	8.27E−15	x	4
		Algorithm 5	800.00	6.27E−12	x	
		Algorithm 6	1773.00	1.14E−14	x	
		SSM	−18161.00	1.21E−23	√	
Algorithm 4	2491.00 (± 66.74)	Algorithm 5	−578.00	1.20E−12	√	2
		Algorithm 6	395.00	4.54E−11	x	
		SSM	−19539.00	1.91E−20	√	
Algorithm 5	3069.00 (±78.52)	Algorithm 6	973.00	1.90E−15	x	3
		SSM	−18961.00	5.01E−21	√	
Algorithm 6	2096.00 (±48.18)	SSM	−19934.00	1.32E−19	√	1
SSM	22030.00 (±254.08)				x	7

Table 7
Sampling rate comparison of Algorithm 1 and SSM for Graph 1.

Convergence rate $(1 - \varepsilon)$	Learning rate	Sampling rate	
		Algorithm 1	SSM
0.50	0.1343	101.40	3171
0.60	0.1225	110.49	3187
0.70	0.0930	142.27	3118
0.75	0.0859	153.93	3149
0.80	0.0815	166.20	3152
0.85	0.0688	192.62	3261
0.90	0.0620	208.35	3404
0.95	0.0512	257.80	3445
0.99	0.0325	495.69	3628

Algorithm 1 converges to the CDS ω_i. $\underline{q}(k)$ is updated by Algorithm 1. The learning rate of Algorithm 1 needs to be obtained for each error parameter $\varepsilon \in (0,1)$. To do that, parameter x can be obtained from Eq. (20)

$$(1 - e^{-q_i x})/(1 - \varepsilon) = (1 - e^{-x}) \tag{20}$$

for a given error parameter ε. After some simplification, the above equation is rewritten as an equation of degree q_i as follows

$$(e^{-x})^{q_i} + e^{-x}(1 - \varepsilon) - \varepsilon = 0$$

Value x is obtained by solving (through the numerical methods) the above equation. Learning rate a can be determined by solving Eq. (21) based on parameter x

$$ax/e^{ax} - 1 = \max_{j \neq i}(d_j/d_i) \tag{21}$$

Now, Algorithm 1 is run for learning rate a (obtained from Eq. (21)) and its sampling rate is computed. The minimum number of samples taken by the standard sampling method, satisfying $prob[|\bar{x}_n - \mu| < \delta] \geqslant 1 - \varepsilon$, has also been given in Tables 1 and 3 where $\delta = 0.001$. Comparing the results of Algorithm 1 and the standard sampling method, we find that the number of samples taken by Algorithm 1 is much less than that of the standard sampling method. These results are summarized in Table 7.

6. Conclusion

In this paper, we first introduced the stochastic version of the minimum connected dominating set problem in vertex-weighted graphs, where the PDF of the weight associated with the graph vertices is unknown. Then, we proposed six learning automata-based algorithms for solving the problem. Algorithm 1 guaranteed neither the domination of all the graph vertices nor the connection of the dominators. Therefore, it took a large number of unnecessary samples. Algorithms 2 and 3 were proposed to resolve the domination problem in Algorithm 1. Algorithm 4 that is an improvement of Algorithm 3 avoids sampling of the dominators by which no more vertices is dominated. Algorithm 5 solves both the connection and domination problems of Algorithm 1, and Algorithm 6 improves the performance of Algorithm 5 by reducing the CDS size and unnecessary sampling rate. The convergence of Algorithm 1 to the optimal solution was proved based on the Martingale Theorem. We also studied the relationship between the learning rate and the convergence rate of Algorithm 1. The obtained results confirmed that by a proper choice of the learning parameters, the probability of finding the minimum CDS is close enough to unity. Proper choice of the learning parameters greatly depends on the nature of the application in which the MCDS algorithm is used. If an application sacrifices the solution optimality in favor of the algorithm cost (for example time complexity), the learning rate can be chosen large enough. Otherwise, if the optimality of the solution is more important, learning rate should be selected as small as possible. In most applications, a trade-off between the cost and optimality is desired. To achieve a trade-off for an application, the proposed method first determines the error rate by which the application requirements are met. Then, using Theorem 3, the learning rate by which the given error rate is satisfied is computed. To show the performance of the proposed algorithms, we conducted several experiments on well-known benchmark graphs as well as random generated geometric graphs. Computer simulation showed that the sampling process used in the proposed algorithms focuses on the vertices that form the minimum CDS as the algorithm proceed. Numerical results showed that the sampling rate of Algorithm 1 is much less than that of standard sampling method to obtain the same confidence level.

Appendix A. Standard sampling method

Theorem 7.1. *To obtain a confidence level not smaller than $1-\varepsilon$ for the CDS, it is sufficient to build a confidence with level $1-\varepsilon_i$ for every vertex v_i such that $\sum_{i=1}^{k}\varepsilon_i=\varepsilon$, where k denotes the cardinality of the CDS.*

Proof. The sample size required for each vertex v_i to satisfy a confidence level $1-\varepsilon_i$ is obtained by using the vertex sampling method described below.

Vertex sampling method. Let $(x_1,x_2,\ldots,x_N)$ be a sample of the random variable X having unknown mean μ and variance σ^2. If $\bar{x}\pm\sigma/\sqrt{N\varepsilon_i}$ (where $\bar{x}=\frac{1}{N}\sum_{j=1}^{N}x_j$) is a $1-\varepsilon_i\%$ confidence interval for mean μ, then there exists a positive number N_0 for each sufficiently small value of δ such that

$$p\{|\bar{x}_N-\mu|\prec\delta\}\succ 1-\varepsilon_i \tag{A.1}$$

for all $N\geqslant N_0$.

The problem is to find a confidence interval for each vertex under which a desired confidence level $1-\varepsilon$ is guaranteed for the CDS. The confidence interval for the CDS is defined as the intersection $\overset{k}{\underset{i=1}{\cap}}C_i(\varepsilon_i)$ of the confidence intervals for the vertices, where $C_i(\varepsilon_i) = 1-\varepsilon_i$ denotes the confidence interval for vertex v_i and k denotes the cardinality of the CDS. Using Booles-Bonferroni inequality [2], we have

$$\underset{1\leqslant i\leqslant k}{Min}(1-\varepsilon_i)>p\left(\mu_i\in\overset{k}{\underset{i=1}{\cap}}C_i(\varepsilon_i)\right)\geqslant 1-\sum_{i=1}^{k}p[\mu_i\notin C_i(\varepsilon_i)] \tag{A.2}$$

and so

$$\underset{1\leqslant i\leqslant k}{Min}(1-\varepsilon_i)>p\left(\mu_i\in\overset{k}{\underset{i=1}{\cap}}C_i(\varepsilon_i)\right)\geqslant 1-\sum_{i=1}^{k}\varepsilon_i \tag{A.3}$$

Hence, the confidence level of the CDS is not smaller than $1-\sum_{i=1}^{k}\varepsilon_i$. In this theorem, the objective is to obtain a confidence level not smaller than $1-\varepsilon$ for the CDS. To achieve this, according to the Bonferroni Correction [10], it is sufficient to build a confidence with level $1-\varepsilon_i$ for each vertex v_i such that $\sum_{i=1}^{k}\varepsilon_i=\varepsilon$. □

References

[1] J. Akbari Torkestani, M.R. Meybodi, Approximating the minimum connected dominating set in stochastic graphs based on learning automata, in: Proceedings of International Conference on Information Management and Engineering (ICIME 2009), Malaysia, 2009, pp. 672–676.
[2] F.B. Alt, Bonferroni inequalities and intervals, in: Encyclopedia of Statistical Sciences, vol. 1, 1982, pp. 294–301.
[3] K.M. Alzoubi, X.-Y. Li, Y. Wang, P.J. Wan, O. Frieder, Geometric spanners for wireless ad hoc network, IEEE Transactions on Parallel and Distributed Systems 14 (4) (2003) 408–421.
[4] K.M. Alzoubi, P.J. Wan, O. Frieder, Distributed heuristics for connected dominating sets in wireless ad hoc networks, Journal of Communications and Networks 4 (1) (2002) 22–29.
[5] K.M. Alzoubi, P.J. Wan, O. Frieder, New distributed algorithm for connected dominating set in wireless ad hoc networks, in: Proceedings of the 35th Hawaii International Conference on System Sciences, 2002.

[6] K.M. Alzoubi, P.-J. Wan, O. Frieder, Maximal independent set, weakly connected dominating set, and induced spanners for mobile ad hoc networks, International Journal of Foundations of Computer Science 14 (2) (2003) 287–303.
[7] S. Basagni, D. Bruschi, I. Chlamtac, A mobility-transparent deterministic broadcast mechanism for ad hoc network, IEEE/ACM Transactions on Networking 7 (6) (1999) 799–807.
[8] S. Basagni, M. Conti, S. Giordano, I. Stojmenovic, Mobile Ad Hoc Networking, IEEE Press, 2004.
[9] J. Akbari Torkestani, M.R. Meybodi, A link stability-based multicast routing protocol for wireless mobile ad hoc networks, Journal of Network and Computer Applications 34 (4) (2011) 1429–1440.
[10] C.E. Bonferroni, Teoria Statistica Delle Classi e Calcolo Delle Probabilit'a, Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze, vol. 8, 1936, pp. 3–62.
[11] O. Booij, Z. Zivkovic, B. Kröse, Efficient data association for view based SLAM using connected dominating sets, Robotics and Autonomous Systems 57 (12) (2009) 1225–1234.
[12] S. Butenko, X. Cheng, C. Oliveira, P.M. Pardalos, A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks, in: Recent Developments in Cooperative Control and Optimization, Kluwer Academic Publishers, 2004, pp. 61–73.
[13] X. Cheng, M. Ding, D. Hongwei, X. Jia, Virtual backbone construction in multihop ad hoc wireless networks, Journal of Wireless Communications and Mobile Computing 6 (2006) 183–190.
[14] I. Chlamtac, S. Kutten, Tree-based broadcasting in multihop radio networks, IEEE Transactions on Computing 36 (10) (1987) 1209–1223.
[15] I. Chlamtac, M. Conti, J. Liu, Mobile ad hoc networking: imperatives and challenges, Journal of Ad Hoc Networks 1 (2003) 13–64.
[16] B.N. Clark, C.J. Colbourn, D.S. Johnson, Unit disk graphs, Discrete Mathematics 86 (1990) 165–177.
[17] F. Dai, J. Wu, An extended localized algorithm for connected dominating set formation in ad hoc wireless networks, IEEE Transactions on Parallel and Distributed Systems 15 (10) (2004) 908–920.
[18] O. Dousse, F. Baccelli, P. Thiran, Impact of interferences on connectivity in ad hoc networks, IEEE/ACM Transactions on Networking 13 (2) (2005) 425–436.
[19] R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum weight spanning trees, ACM Transaction on Programming Languages and Systems 5 (1983) 66–77.
[20] S. Guha, S. Khuller, Approximation algorithms for connected dominating sets, Algorithmica 20 (4) (1998) 374–387.
[21] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (1979) 65–70.
[22] B. Han, Zone-based virtual backbone formation in wireless ad hoc networks, Ad Hoc Networks 7 (1) (2009) 183–200.
[23] Http://dimacs.rutgers.edu/Challenges.
[24] K.R. Hutson, D.R. Shier, Minimum spanning trees in networks with varying edge weights, Annals of Operations Research, vol. 146, Springer, 2006, pp. 3–18.
[25] S. Lakshmivarahan, M.A.L. Thathachar, Bounds on the convergence probabilities of learning automata, IEEE Transactions on Systems, Man, and Cybernetics 6 (1976) 756–763.
[26] D. Li, H. Du, P.-J. Wan, X. Gao, Z. Zhang, W. Wu, Construction of strongly connected dominating sets in asymmetric multihop wireless networks, Theoretical Computer Science 410 (8–10) (2009) 661–669.
[27] Y. Li, M.T. Thai, F. Wang, C.W. Yi, P.J. Wang, D.Z. Du, On greedy construction of connected dominating sets in wireless networks, Special issue of Wireless Communications and Mobile Computing (WCMC), 2005.
[28] H. Lim, C. Kim, Flooding in wireless ad hoc networks, Journal of Computer Communications 24 (2001) 353–363.
[29] D. Lokshtanov, M. Mnich, S. Saurabh, A linear kernel for a planar connected dominating set, Theoretical Computer Science 412 (23) (2011) 2536–2543.
[30] M.V. Marathe, H. Breu, H.B. Hunt, S.S. Ravi, D.J. Rosenkrantz, Simple heuristics for unit disk graphs, Networks 25 (1995) 59–68.
[31] P. Mohapatra, S. Krishnamurthy, Ad hoc Networks: Technologies and Protocols, Springer Science, 2005.
[32] K.S. Narendra, M.A.L. Thathachar, Learning Automata: An Introduction, Printice-Hall, New York, 1989.
[33] C. Pang, R. Zhang, Q. Zhang, J. Wang, Dominating sets in directed graphs, Information Sciences 180 (2010) 3647–3652.
[34] M.A.L. Thathachar, B.R. Harita, Learning automata with changing number of actions, IEEE Transactions on Systems, Man, and Cybernetics SMG17 (1987) 1095–1100.
[35] Y.-T. Tsai, Y.-L. Lin, F.R. Hsu, Efficient algorithms for the minimum connected domination on trapezoid graphs, Information Sciences 177 (12) (2007) 2405–2417.
[36] V. Turau, B. Hauck, A self-stabilizing algorithm for constructing weakly connected minimal dominating sets, Information Processing Letters 109 (14) (2009) 763–767.
[37] Y. Wang, W. Wang, X.-Y. Li, Distributed low-cost backbone formation for wireless ad hoc networks, in: Proceedings of the Sixth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2005), 2005, pp. 2–13.
[38] J. Wu, H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, in: Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication, 1999, pp. 7–14.
[39] J. Wu, F. Dai, M. Gao, I. Stojmenovic, On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks, Journal of Communications and Networks 4 (1) (2002) 1–12.
[40] J. Wu, B. Wu, I. Stojmenovic, Power-aware broadcasting and activity scheduling in ad hoc wireless networks using connected dominating sets, Journal of Wireless Communications and Mobile Computing 3 (2003) 425–438.
[41] L. Wu, E. Shan, Z. Liu, On the k-tuple domination of generalized de Brujin and Kautz digraphs, Information Sciences 180 (2010) 4430–4435.
[42] R. Xie, D. Qi, Y. Li, J.Z. Wang, A novel distributed MCDS approximation algorithm for wireless sensor networks, Journal of Wireless Communications and Mobile Computing, 2007.
[43] B. Yin, H. Shi, Y. Shang, An efficient algorithm for constructing a connected dominating set in mobile ad hoc networks, Journal of Parallel and Distributed Computing 71 (1) (2011) 27–39.
[44] F. Zou, Y. Wang, X.-H. Xu, X. Li, H. Du, P. Wan, W. Wu, New approximations for minimum-weighted dominating sets and minimum-weighted connected dominating sets on unit disk graphs, Theoretical Computer Science 412 (3) (2011) 198–208.
[45] J. Akbari Torkestani, An adaptive learning automata-based ranking function discovery algorithm, Journal of Intelligent Information Systems, http://dx.doi.org/10.1007/s10844-012-0197-4.
[46] J. Akbari Torkestani, A new approach to the job scheduling problem in computational grids, Journal of Cluster Computing (2012), http://dx.doi.org/10.1007/s10586-011-0192-5.