Learning Automata-Based Algorithms for Finding Minimum Weakly Connected Dominating Set in Stochastic Graphs

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

Mohammad Reza Meybodi
Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, Iran
mmeybodi@aut.ac.ir

Abstract

A weakly connected dominating set (WCDS) of graph G is a subset of G so that the vertex set of the given subset and all vertices with at least one endpoint in the subset induce a connected sub-graph of G. Finding the WCDS is a new promising approach for clustering the wireless networks. The minimum WCDS (MWCDS) problem is known to be NP-hard, and several approximation algorithms have been proposed for solving MWCDS in deterministic graphs. However, to the best of our knowledge no work has been done on finding the WCDS in stochastic graphs. In this paper, a definition of the MWCDS problem in a stochastic graph is first presented and then several learning automata-based algorithms are proposed for solving the stochastic MWCDS problem where the probability distribution function of the weight associated with the graph vertices is unknown. Taking advantage of learning automata, the proposed algorithms significantly reduce the number of samples needs to be taken from the vertices of the stochastic graph. It is shown that by a proper choice of the parameters of the proposed algorithms, the probability of finding the MWCDS is as close to unity as possible. To evaluate the proposed algorithms, the number of samples taken from the graph is compared with that of the standard sampling method (SSM). Experimental results show the major superiority of the proposed algorithms over the SSM in terms of the sampling rate.

Keyword Weakly connected dominating set, stochastic graph, learning automata, distributed learning automata.

1. Introduction

The dominating set (DS) problems are a class of optimization problems which are widely used in wireless ad hoc networks [1, 3-4, 7-14, 25-27]. The minimum connected dominating set (MCDS) of the network topology graph forms a virtual backbone by which the routing overhead can be significantly reduced, where the number of vertices responsible for the route discovery and data transmission can be reduced to the number of vertices in the MCDS [9, 12, 23-27]. A WCDS of a graph G is a dominating subset

of G so that the vertex set of the given subset and all the vertices with at least one endpoint in the subset induce a connected sub-graph of G. Finding the WCDS [1, 3, 7-9, 11, 13] is a well-known approach for clustering the wireless ad hoc networks [22, 28-35], and was first proposed by Chen and Listman in [7]. In this clustering method, the dominator nodes and their closed neighbors assume the role of the cluster-heads and cluster members, respectively. The structure of the network graph can be simplified using WCDS and made more succinct for routing in ad hoc networks [12, 14]. Dunbar et al. [36] studied weakly connected domination in graphs and showed that the problem of finding a minimum size WCDS in a given graph is NP-Hard.

Chen and Listman [7] first proposed a distributed approximation algorithm for clustering the wireless ad hoc networks by finding a near optimal solution to the minimum size WCDS problem. The proposed algorithm is also inspired by Guha and Khuller's centralized approximation algorithm [11] for finding small connected dominating sets (CDS). Guha and Khuller [11] proposed two centralized greedy heuristic algorithms with bounded performance guarantees for connected dominating set formation. In the first algorithm, the connected dominating set is grown from one node outward, and in the second algorithm, a WCDS is constructed, and then the intermediate nodes are selected to create a CDS. Chen and Liestman also proposed a zonal algorithm [1, 8] in which the graph is divided into regions. A WCDS is constructed for each region, and adjustments are made along the borders of the regions to produce a WCDS for the whole graph. Their algorithm for the partitioning phase is partly based on the algorithm proposed by Gallager et al. [10] for finding the minimum spanning tree in weighted graphs. Han and Jia [3] also proposed an area-based distributed algorithm for WCDS construction in vertex-weighted graphs with constant approximation ratio, linear time and message complexity. While it has a lower message complexity than the zonal algorithm proposed by Chen and Listman, it outperforms the mentioned algorithm. Alzoubi et al. [9] presented two distributed algorithms for finding a WCDS in ad hoc networks. The first algorithm was implemented by first electing a leader among the nodes, which was going to be the root of a spanning tree. The spanning tree is then traversed and the dominator nodes are selected. But the distributed leader election is extremely expensive in practice, and exhibits a very low degree of parallelism. The second algorithm first constructs a maximum independent set (MIS) by an iterative labeling strategy, and then modifies the formed MIS by selecting one intermediate node between each pair of dominators separated by exactly three hops.

In all existing methods proposed for solving the minimum WCDS problem, either the WCDS is constructed for unweighted graphs or it is assumed that the graph is deterministic and so the weight associated with the graph vertices is fixed and does not vary with time. Since the characteristics of the wireless ad hoc networks are stochastic, unpredictable and time-varying [2, 37-38], such an assumption can not hold true in realistic applications. Therefore, the stochastic graphs (which are described later in Subsection 2.2) are more appropriate structures for modeling the realistic behaviors and dynamics of the ad hoc networks. The minimum WCDS problem in deterministic graphs has been widely studied in the literature and a host of solutions are available [1, 3, 7-9, 11, 13], but to the best of our knowledge, the idea of solving the minimum WCDS problem in a stochastic weighted graph is discussed for the first time here. In this paper, the stochastic minimum weakly connected dominating set problem is first introduced and then five learning automata-based approximation algorithms are proposed for solving the mentioned problem, when the probability distribution function (PDF) of the weight of the vertices is unknown. Each of the proposed algorithms is composed of the several stages, and the vertices that must be sampled at each stage are randomly selected by the learning automata. As the proposed algorithms proceed, the sampling process is concentrated on the vertices that construct the WCDS with the minimum expected weight. Hence, the proposed learning automata-based algorithms are able to reduce the number of samples needs to be taken from the graph by reducing the rate of unnecessary samples taken from the

vertices which are not included in the MWCDS. To show the performance of the proposed algorithms, we have conducted two groups of experiments on stochastic graphs. In the first group, the proposed algorithms are tested on several stochastic benchmark graphs. The second group of experiments investigates the scalability of the proposed algorithms on a subset of dense stochastic graphs. In all experiments conducted in this paper, the number of samples taken by the proposed algorithms from the stochastic graphs is compared with that of the standard sampling method. Experimental results show the huge superiority of the proposed algorithms over the SSM in terms of the sampling rate.

The rest of the paper is organized as follows. In the next section, the dominating sets, stochastic graphs, and learning automata are briefly introduced. The proposed learning automata-based algorithms are described in Section 3. In Section 4, the convergence results of the first proposed algorithm are studied. In this section, strong convergence theorems in which the convergence of the first proposed algorithm to the optimal solution is proved are first presented. Then, the relationship between the learning rate and the convergence rate of algorithm is studied. The performance of the proposed algorithms is evaluated through the simulation experiments in section 5. Section 6 concludes the paper.

2. Preliminaries and Definitions

In this section, to provide a sufficient background for the remainder of the paper, we present a brief overview of the dominating set problems and their applications in communication networks, stochastic graph, and learning automata and its variations.

2.1.Dominating Sets

Dominating set S of a given graph $G < V, E >$ is a subset of vertex set V such that every vertex $v \subseteq V$ is either in S or adjacent to a vertex of S. A vertex of S is said to dominate itself and all adjacent vertices. Each vertex is called a dominator, if it is in the dominating set, otherwise it is called a dominatee. That is, each dominatee is dominated by at least one dominator. A minimum DS (MDS) is a DS with the minimum cardinality. A dominating set is also an independent dominating set, if no two vertices in the set are adjacent. A CDS S of a given graph G is a dominating set whose induced sub-graph, denoted$< S >$, is connected, and a MCDS is a CDS with the minimum cardinality. A MCDS forms a virtual backbone in the graph by which the routing overhead can be significantly reduced. The MDS and MCDS problems have been shown to be NP-Hard [5, 6], and even for a unit disk graph (UDG), the problem of finding a MCDS is still NP-Hard [6].

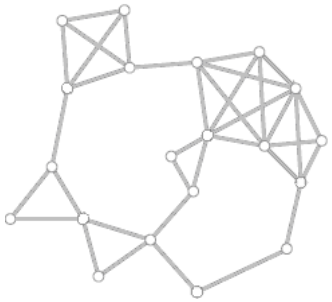


Figure 1.A. A sample unit disk graph

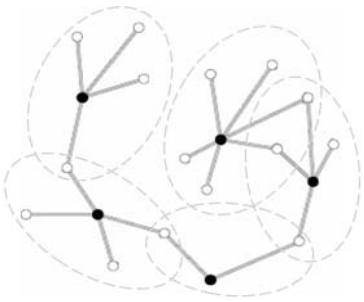


Figure 1.B. The weakly connected dominating set

The closed neighborhood $N_G[v]$ of a vertex v in graph G consists of the vertices adjacent to v and vertex v itself. The closed neighborhood $N_G[S]$ of the set S is the union $\cup_{v \in S} N_G[v]$. The subscript G can be omitted if the meaning is clear from the context. A dominating set S is a WCDS of a graph G, if the graph $< S >_W = \big(N[S], E \cap (N[S] \times S)\big)$ is a connected sub-graph of G. In other words, the weakly induced sub-graph $< S >_W$ contains the vertices of S and all edges of G which have at least one end point in S. Finding the minimum WCDS in the UDG is a NP-Hard problem [3, 36] and is one of the most investigated methods for cluster formation, in which a dominator node assumes the role of a cluster-head and its one-hop neighbors are assumed to be cluster members. A sample UDG and one of its WCDS are shown in Figures 1.A and 1.B, respectively. The dominator nodes assume the role of the cluster-heads and they have been colored black.

2.2.Stochastic Graph

A vertex-weighted graph can be described by a triple $G < V, E, W >$, where $V = \{v_1, v_2, \dots, v_n\}$ denotes the vertex set, $E \subseteq V \times V = \{e_1, e_2, \dots, e_m\}$ denotes the edge set and $W = \{w_1, w_2, \dots, w_n\}$ denotes the set of vertex weights such that $W(v_i) = w_i$ (for all $i \in \{1,2, \dots, n\}$). In most scenarios, the weight of the vertices is assumed to be fixed, but this is not always true and it varies with time. A vertex-weighted graph is a stochastic vertex-weighted graph, and hereafter in this paper is called stochastic graph, if the weight of vertex v_i, for all $i \in \{1,2, \dots, n\}$, is a random variable with probability distribution function w_i. Let $G < V, E, W >$ and $\mathcal{S} = \{\delta_1, \delta_2, \dots, \delta_r\}$ be a stochastic graph and the set of all its WCDSs, respectively. If $\overline{W}(v_i)$ and $\overline{W}(\delta_i) = \sum_{v_i \in \delta_i} \overline{W}(v_i)$ denote the expected weight of vertex v_i and the expected weight of the WCDS δ_i, respectively, therefore the stochastic WCDS $\delta^* \in \mathcal{S}$ of graph G is minimum, if and only if $\overline{W}(\delta^*) = \min_{\forall \delta_i \in \mathcal{S}} \overline{W}(\delta_i)$. That is, the minimum WCDS of a given stochastic graph G is defined as the stochastic WCDS with the minimum expected weight.

2.3.Learning Automata

A learning automaton [15-21] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized. Figure 2 shows the relationship between the learning automaton and random environment.

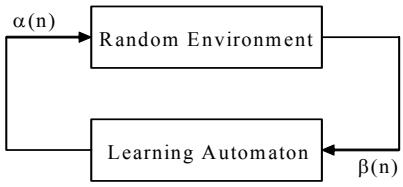


Figure 2. The relationship between the learning automaton and its random environment

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, \dots, \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, \dots, \beta_m\}$ denotes the set of the values that can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, \dots, c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary environment. The environments depending on the nature of the reinforcement signal $\underline{\beta}$ can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval [0, 1] can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval $[a, b]$.

Learning automata can be classified into two main families [15]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $< \underline{\beta}\underline{\alpha}, T >$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and T is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha(k)$ and $\underline{p}(k)$ denote the action chosen at instant k and the action probability vector on which the chosen action is based, respectively. The recurrence equation shown by (3) and (4) is a linear learning algorithm by which the action probability vector p is updated. Let $\alpha_i(k)$ be the action chosen by the automaton at instant k.

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

when the taken action is rewarded by the environment (i.e., $\beta(n) = 0$) and

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

when the taken action is penalized by the environment (i.e., $\beta(n) = 1$), r is the number of actions that can be $a(k)$ and $b(k)$ denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. If $a(k) = b(k)$, the recurrence equations (3) and (4) are called linear reward-penalty (L_{R-P}) algorithm, if $a(k) \gg b(k)$ the given equations are called linear reward-ϵpenalty ($L_{R-\epsilon P}$), and finally if $b(k) = 0$ they are called linear reward-Inaction (L_{R-I}). In the latter case, the action probability vectors remain unchanged when the taken action is penalized by the environment.

2.3.1. Distributed Learning Automata

A Distributed learning automata (DLA) [21] shown in Figure 3 is a network of interconnected learning automata which collectively cooperate to solve a particular problem. Formally, a DLA can be defined by a quadruple $< A, E, T, A_0 >$, where $A = \{A_1, A_2, \dots, A_n\}$ is the set of learning automata, $E \subset A \times A$ is the set of the edges in which edge $e_{(i,j)}$ corresponds to the action α_{ij} of the automaton A_i, T is the set of learning schemes with which the learning automata update their action probability vectors, and A_0 is the root automaton of DLA from which the automaton activation is started.

The operation of a DLA can be described as follows: At first, the root automaton randomly chooses one of its outgoing edges (actions) according to its action probabilities and activates the learning

automaton at the other end of the selected edge. The activated automaton also randomly selects an action which results in activation of another automaton. The process of choosing the actions and activating the automata is continued until a leaf automaton (an automaton which interacts to the environment) is reached. The chosen actions, along the path induced by the activated automata between the root and leaf, are applied to the random environment. The environment evaluates the applied actions and emits a reinforcement signal to the DLA. The activated learning automata along the chosen path update their action probability vectors on the basis of the reinforcement signal by using the learning schemes. The paths from the unique root automaton to one of the leaf automata are selected until the probability with which one of the paths is chosen is close enough to unity. Each DLA has exactly one root automaton which is always activated, and at least one leaf automaton which is activated probabilistically.

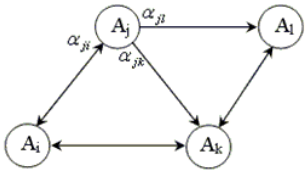


Figure 3. Distributed learning automata

2.3.2. 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 [17] that a learning automaton with a changing number of actions is absolutely expedient and also ϵ-optimal, when the reinforcement scheme is L_{R-I}. Such an automaton has a finite set of n actions, $\underline{\alpha} = \{\alpha_1, \alpha_2, \dots, \alpha_r\}$. $A = \{A_1, A_2, \dots, A_m\}$ denotes the set of action subsets and $A(k) \subseteq \alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $\Psi(k) = \{\Psi_1(k), \Psi_2(k), \dots, \Psi_m(k)\}$ defined over the possible subsets of the actions, where $\Psi_i(k) = prob[A(k) = A_i | A_i \in A, 1 \leq i \leq 2^n - 1]$.

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

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

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 probabilities in a variable action-set learning automaton can be described as follows. Let $A(k)$be the action subset selected at instant n. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in equation (3). The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the 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 [17].

3. The Proposed Algorithms

In this section, several learning automata-based approximation algorithms are proposed for solving the minimum WCDS problem in stochastic graphs. In the proposed algorithms, the vertex-weighted stochastic graph $G < V, E, W >$ is served as the input of algorithm and the output is the WCDS of G. The reinforcement scheme by which the proposed algorithms update the probability vectors is L_{R-I}. It is assumed that the weight associated with each vertex of the stochastic graph G is a random variable with unknown probability distribution function (PDF). Therefore, the PDF of the random variables must be estimated by a statistical method. The proposed algorithms taking advantage of learning automata are able to precisely determine the weight distribution of the vertices with a small number of samples. In these algorithms, a network of the learning automata isomorphic to the stochastic graph G is initially formed by assigning to each vertex v_i of the graph learning automaton A_i. The resultant network of learning automata can be described by a triple $< \underline{A}, \underline{\alpha}, \underline{W} >$, where $\underline{A} = \{A_1, A_2, \dots, A_n\}$ denotes the set of the learning automata, $\underline{\alpha} = \{\underline{\alpha_1}, \underline{\alpha_2}, \dots, \underline{\alpha_n}\}$ denotes the set of actions in which $\underline{\alpha_i} = \{\alpha_i^1, \alpha_i^2, \dots, \alpha_i^{r_i}\}$ defines the set of actions that can be taken by learning automaton A_i, for each $\underline{\alpha_i} \in \underline{\alpha}$, and $\underline{W} = \{w_1, w_2, \dots, w_n\}$ denotes the set of weights such that w_i (for all $i \in \{1,2, \dots, n\}$) is the random weight associated with automaton A_i.

In the first proposed algorithm, the action-set of each learning automaton A_i, referred to as $\underline{\alpha_i}$, contains two actions α_i^0 and α_i^1. Choosing action α_i^0 by learning automaton A_i declares vertex v_i as a dominatee vertex and choosing action α_i^1 declares it as a dominator vertex. The proposed algorithm consists of a number of stages, and at each stage the vertices which are selected as dominators are added to the set of vertices by which a WCDS may be formed in that stage. This algorithm iteratively constructs a number of WCDSs and updates the action probability vectors until it finds a near optimal solution to the minimum WCDS problem. Stage k of the proposed algorithm is briefly described in the following steps:

Step 1. WCDS formation

While the cardinality of the dominatee set is less than the cardinality of vertex set V do
  For all learning automata do in parallel
    Each automaton A_i chooses one of its 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 dominators
      Vertex v_i and all its neighbors are added to the dominatee set
      The random weight associated with vertex v_i is added to the weight of dominator set

Step 2. Comparing the weight of the constructed WCDS with a dynamic threshold

Let us assume that WCDS δ_i is selected at stage k.

The average weight of the constructed WCDS δ_i until stage k is computed as

$$\overline{W}(\delta_i^k) = \frac{1}{k_i} \sum_{j=1}^{k_i} W(\delta_i(j)) \quad (4)$$

where k_i denotes the number of times the WCDS δ_i is constructed until stage k, and $W(\delta_i(j))$ denotes the weight of the j^{th} sampled WCDS δ_i which is defined as

$$W(\delta_i(j)) = \sum_{v_s \in \delta_i} W(v_s(j)) \quad (5)$$

where $W(v_s(j))$ denotes the weight of the vertex v_s in the j^{th} sample taken from δ_i.

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

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

where r denotes the number of all WCDSs of G.

Step 3. Updating the action probability vectors

Depending on the comparison result in step 2, all the learning automata (only the activated learning automata in Algorithm 5) reward their chosen actions if the average weight of the constructed WCDS δ_i is less than or equal to the dynamic threshold T_{k-1}, and penalize them otherwise. Each learning automaton then updates its action probability vector using L_{R-I} reinforcement scheme.

Step 4. Stopping Condition

The process of constructing the WCDSs and updating the action probabilities are repeated until the product of the probability of choosing the vertices of the constructed WCDS is greater than a certain threshold (P) or the number of constructed WCDS exceeds a pre-specified threshold (K). The WCDS which is formed just before the algorithm stops is the WCDS with the minimum expected weight among all WCDSs of the stochastic graph G.

The proposed algorithm has been described in Figure 4 in more detail.

Algorithm 1 The first proposed algorithm for solving the Stochastic MWCDS problem

Input Stochastic Vertex-Weighted Graph $G < V, E, W >$, Thresholds K, P
Output The minimum weakly connected dominating set
Assumptions
  Assign a learning automaton A_i to each vertex v_i
  Let $\underline{\alpha_i} = \{\alpha_i^0, \alpha_i^1\}$ denotes the action-set of learning automaton A_i
Begin
  Let T_k be the dynamic threshold at stage k initially set to n
  Let $\overline{W}(\delta_i^k)$ be the average weight of all WCDSs δ_i sampled until stage k, and initially set to 0
  Let k denotes the stage number and is initially set to 0
  Repeat
    Let d_k denotes the dominatee set constructed at stage k and is initially set to null
    Let δ_i^k be the dominator set δ_i constructed at stage k and initially set to null
    Let ω_k be the weight of the WCDS constructed at stage k and initially set to 0
    While $|d_k| < n$ do
      For all the vertices of the stochastic graph do in parallel
        Learning automaton A_i randomly chooses one of its actions
        If the chosen action is α_i^1 (declares vertex v_i as a dominator) **Then**
          Vertex v_i is added to the dominator set δ_i^k
          Vertex v_i and all its neighbors are added to the dominatee set d_k
          The random weight associated with the selected action is added to ω_k
        End If
      End For
    End While
    Compute $\overline{W}(\delta_i^k)$ as the average weight of all sampled WCDS δ_i^k until stage k by Equation (4)
    If the average weight $\overline{W}(\delta_i^k)$ is less than the dynamic threshold T_{k-1} **Then**
      The chosen actions by all the learning automata are rewarded
    Else
      The chosen actions by all the learning automata are penalized
    End If
    Compute T_k as the average weight of all the WCDSs constructed until stage k by Equation (6)
    $k \leftarrow k + 1$
  Until the probability of constructing WCDS δ_i^k is greater than P or stage number k exceeds K
End Algorithm

Figure 4. Pseudo code of Algorithm 1

In Algorithm 1, it is assumed that the number of actions that can be taken by each learning automaton is fixed and does not vary with time. Fixed action-set learning automaton used in Algorithm 1 significantly increases the number of vertices that must be sampled for constructing the WCDS (or the cardinality of the constructed WCDS) and hence increases the running time of algorithm dramatically. To alleviate this problem, we propose another algorithm called *Algorithm 2* in which the number of actions available for each learning automaton changes at each stage. The action-set of Algorithm 2 is similar to that of Algorithm 1. In this algorithm, each learning automaton A_i disables its action α_i^1 (which declares vertex v_i as a dominator vertex) as described in Subsection 2.3.2 on variable action-set learning automata, if one of its neighbors has declared itself as a dominator vertex earlier. The action-set remains unchanged otherwise. In this algorithm, the learning automata are randomly and sequentially activated and choose one of their available actions based on the action probability vector scaled after disabling the actions as given in equation (3). The action probabilities are then rescaled after evaluating (rewarding or penalizing) the constructed WCDS. This reduction in number of actions considerably reduces the cardinality of the constructed WCDS and so increases the speed of convergence to the minimum WCDS.

Since the first two proposed algorithms (Algorithms 1 and 2) can not guarantee to from a WCDS at each stage, the first step (i.e., WCDS formation step) of these algorithms needs to be repeated until a weakly connected dominating set is finally formed. This dramatically increases the running time of algorithm. To solve the above mentioned problem, we propose another learning automata-based algorithm in which a weakly connected dominating set can be constructed at each stage. This algorithm which is called *Algorithm 3*, is very similar to Algorithm 1, but differs in the action-set defined for each learning automaton as well as the way of constructing the WCDSs (step 1). In Algorithm 3, learning automaton A_j (for all $j \in \{1,2,\dots,n\}$) assigned to vertex v_j, where vertex v_j is adjacent to vertex v_i, as well as learning automaton A_i itself, form the action-set of learning automaton A_i. That is, $\underline{\alpha_i} = \{A_j \mid \forall j; j = i \text{ or } (v_i, v_j) \in E\}$. For each learning automaton A_i, choosing a given action $A_j \in \underline{\alpha_i}$ declares vertex v_j (the vertex corresponding to the selected action) as its dominator. In other words, each learning automaton chooses a dominator for its corresponding vertex among its neighbors or declares itself as a dominator node. The dominators selected by all learning automata constitute the current WCDS. The step of WCDS formation of Algorithm 3 (step 1) at each stage k is described as shown in Figure 5. The remaining steps (*steps 2, 3, and 4*) of Algorithm 3 are the same as those described in Algorithm 1.

Algorithm 3 The first step of the third proposed stochastic MWCDS algorithm

01:**For** all learning automata do in parallel
02: Automaton A_i chooses one of its actions (e.g. action A_j)
03: The vertex corresponding to the selected action (i.e., vertex v_j) is added to δ_i^k
04: The random weight associated with the selected action is added to ω_k
05:**End For**

Figure 5. The WCDS formation step in Algorithm 3

The dominating set formation process of Algorithm 3 may result in redundant dominators in WCDS by which no more vertices can be dominated. This enlarges the size of WCDS and increases the sampling rate of algorithm too. As shown in Algorithm 2, the variable action-set learning automata enable algorithm to reduce the number of vertices needs to be sampled (sampling rate) for constructing the WCDS at each stage. The reduced sampling rate increases the convergence speed to the optimal solution. To achieve such a performance in Algorithm 3, we propose another algorithm in which the action-set of each learning automaton changes with time as follows:

if vertex v_i is selected as a dominator, then learning automaton A_i and all the learning automata which are adjacent to learning automaton A_i retain the action corresponding to learning automaton A_i if any, and disable all the other actions (if they have not been disabled yet) as described in Subsection 2.3.2. That is, when a given vertex v_i (associated with automaton A_i) is selected as a dominator, the action-set of its neighboring learning automata is pruned in such a way that the dominator that can be chosen by learning automaton A_i and its neighbors is vertex v_i only. Such a pruned action-set avoids selecting the adjacent vertices as dominators, and so causes significantly smaller WCDSs. In this algorithm, the learning automata are sequentially activated at random, and each learning automaton chooses one of its possible actions according to its scaled action probability vector as described in Algorithm 2. The algorithm in which the WCDS is thus constructed is called *Algorithm 4*.

The last algorithm we propose for approximating a near optimal solution to the stochastic minimum WCDS problem is based on distributed learning automata (DLA) described in Subsection 2.3.1 which is called *Algorithm 5*. This algorithm like Algorithm 3 differs from Algorithm 1 only in the action-set defined for each learning automaton as well as the method by which the WCDS is constructed at each stage. In this algorithm, the set of actions that can be taken by each learning automaton A_i initially includes the set of all learning automata (i.e., $\underline{A} = \{A_1, A_2, \ldots, A_n\}$) except automata A_i and its neighbors. That is, the action-set of learning automaton A_i is defined as $\underline{\alpha_i} = \underline{A} - (\{A_i\} \cup \{A_j | \forall j; (v_i, v_j) \in E\})$. Each learning automaton can be in one of two states: active and passive. Each learning automaton is initially set to the passive state. The selected automata at each stage are activated and added to the set of dominator vertices. The WCDS formation step of Algorithm 5 at each stage k is described as shown in Figure 6. The remaining steps (*steps 2, 3, and 4*) of Algorithm 5 are the same as those described in Algorithm 1.

Algorithm 5 The first step of the fifth proposed stochastic MWCDS algorithm

01:The first dominator is selected by a separate automaton whose action-set is the vertex set of G
02:The selected vertex is denoted by v_i and added to δ_i^k
03:**Repeat**
04: Vertex v_i and all its neighbors are added to the dominatee set d_k
05: The random weight of the selected vertex is added to ω_k
06: Automaton A_i disables the actions corresponding to all members of the dominatee set
07: Automaton A_i chooses one of its actions according to its scaled action probability vector
08: The vertex corresponding to the selected action is denoted by v_i and added to δ_i^k
09: Vertex v_i is set to vertex v_j
10:**Until** $|d_k| \geq n$

Figure 6. The WCDS formation step in Algorithm 5

4. Convergence Results

In this section we prove two main results of the paper. The first result is concerned with the proof of the convergence of Algorithm 1 to the optimal solution. The method used to prove the convergence of the Algorithm 1 follows the method given in [15, 19] to analysis the behavior and to prove the convergence of the learning automata operating in non-stationary environments. In addition to the convergence proof, it is also shown that by a proper choice of the learning rate a, the probability with which Algorithm 1 finds the optimal solution is as close to unity as possible where each learning automaton has a L_{R-I} learning algorithm. The second result concerns the relationship between the convergence error parameter ε (i.e., the error parameter involved in the SSM) and the learning rate of Algorithm 1. The second result we

discuss in this paper demonstrates the existence of a learning rate under which the probability of constructing (or converging to) the WCDS with the minimum expected weight is larger than $1-\varepsilon$.

Theorem 1 Let $q_i(k)$ be the probability of constructing the WCDS ω_i in a given stochastic graph, at stage k. If $\underline{q}(k)$ evolves 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

$$Prob[\lim_{k\to\infty} q_i(k) = 1] \geq 1-\varepsilon \tag{7}$$

Proof The steps of the convergence proof are briefly outlined as follows. At first, it is proved that, the penalty probabilities of the WCDSs converge to the constant values of the final penalty probabilities, for large enough values of k (*lemma1*). Then, it is shown that, the probability of choosing the WCDS 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 WCDS is always nonnegative (lemmas 2 and 3). Finally, the convergence of the Algorithm 1 to the minimum WCDS 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 WCDS ω_i is penalized with probability $c_i(k)$ at stage k (i.e., $c_i(k) = Prob\left[\overline{W}_{\omega_i} > T_k\right]$), and $\lim_{k\to\infty} c_i(k) = {c_i}^*$, then for each $\varepsilon \in (0,1)$ and $k > K(\varepsilon)$ we have, $Prob[|{c_i}^* - c_i(k)| > 0\ ] < \varepsilon$.

Proof Let $c_j(k)$ (for all $j = 1,2,\dots,r$) denotes the probability with which the WCDS ω_i is penalized at stage k, r denotes the number of all WCDSs of G, and ${c_j}^*$ denotes the probability $c_j(k)$ for large values of k. Using weak law of large numbers, we conclude that

$$\lim_{k\to\infty} Prob[|{c_i}^* - c_i(k)| > \varepsilon\ ] \to 0 \tag{8}$$

For every $\varepsilon \in (0,1)$, there exists a $a^*(\varepsilon) \in (0,1)$ and $K_0(\varepsilon) < \infty$ such that for all$a < a^*$ and $k > K_0$ we have, $Prob[|{c_i}^* - c_i(k)| > 0\ ] < \varepsilon$, and the proof of lemma 1 is completed. ■

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

$$E[q_i(k+1)|q(k)] = \sum_{j=1}^{r} q_j(k)\left[c_j(k)q_i(k) + d_j(k)\left(\prod_{v_m\notin\omega_i}\left(1-p_1^m(k)\right)\prod_{v_m\in\omega_i} p_1^m(k)\right)\right] \tag{9}$$

where

$$p_1^m(k+1) = \begin{cases} p_1^m(k) + a\left(1-p_1^m(k)\right) & ;\ v_m \in \omega_j \\ p_1^m(k)(1-a) & ;\ v_m \notin \omega_j \end{cases}$$

where $p_1^m(k)$ is the probability that vertex v_m declares itself as a dominator vertex, at stage k, and $v_m \in \omega_i$, if it declares itself as a dominator vertex (or chooses action α_m^1) and $v_m \notin \omega_i$ otherwise.

Proof Since the reinforcement scheme used to update the probability vectors in Algorithm 1 is L_{R-I}, at each stage k, the probability of constructing the WCDS ω_i (i.e., $q_i(k)$), remains unchanged with probability $c_j(k)$ (for all $j = 1,2, \ldots, r$), when the constructed WCDS ω_j is penalized by the random environment. On the other hand, when the constructed WCDS ω_j is rewarded, the probability of choosing the vertices of the WCDS ω_i, which are in the constructed WCDS ω_j increases by a given learning rate as that of the other vertices decreases. To illustrate the proof of the lemma in more details, we prove it for the minimum WCDS of the graph shown in Figure 7. As shown in Figure 7, graph G has 4 vertices, 4 edges, and 11 WCDSs as follows: $\omega_1 = \{v_1\}$, $\omega_2 = \{v_1, v_2\}$, $\omega_3 = \{v_1, v_3\}$, $\omega_4 = \{v_1, v_4\}$, $\omega_5 = \{v_2, v_4\}$, $\omega_6 = \{v_3, v_4\}$, $\omega_7 = \{v_1, v_2, v_3\}$, $\omega_8 = \{v_1, v_2, v_4\}$, $\omega_8 = \{v_1, v_3, v_4\}$, $\omega_{10} = \{v_2, v_3, v_4\}$, $\omega_{11} = \{v_1, v_2, v_3, v_4\}$. It is also assumed that ω_1 is the WCDS with the minimum expected weight of the graph given in Figure 7.

Figure7. Sample graph G and its minimum WCDS

Let

$$q_i(k) = \prod_{v_m \notin \omega_i} p_0^m(k) \prod_{v_m \in \omega_i} p_1^m(k) \tag{10}$$

be the probability of constructing the WCDS ω_i at stage k, where p_0^m and p_1^m denote the probability with which vertex v_m declares itself as a dominatee vertex (or chooses action α_m^0) and as a dominator vertex (or chooses action α_m^1), respectively, and $v_m \in \omega_i$, if it declares itself as a dominator vertex and $v_m \notin \omega_i$ otherwise. Therefore, for the graph shown in Figure 7 we have

$$q_1(k) = p_1^1(k)p_0^2(k)p_0^3(k)p_0^4(k)$$
$$q_2(k) = p_1^1(k)p_1^2(k)p_0^3(k)p_0^4(k)$$
$$q_3(k) = p_1^1(k)p_0^2(k)p_1^3(k)p_0^4(k)$$
$$q_4(k) = p_1^1(k)p_0^2(k)p_0^3(k)p_1^4(k)$$
$$q_5(k) = p_0^1(k)p_1^2(k)p_0^3(k)p_1^4(k)$$
$$q_6(k) = p_0^1(k)p_0^2(k)p_1^3(k)p_1^4(k)$$
$$q_7(k) = p_1^1(k)p_1^2(k)p_1^3(k)p_0^4(k)$$
$$q_8(k) = p_1^1(k)p_1^2(k)p_0^3(k)p_1^4(k)$$
$$q_9(k) = p_1^1(k)p_0^2(k)p_1^3(k)p_1^4(k)$$
$$q_{10}(k) = p_0^1(k)p_1^2(k)p_1^3(k)p_1^4(k)$$
$$q_{11}(k) = p_1^1(k)p_1^2(k)p_1^3(k)p_1^4(k)$$

The conditional expectation of $q_1(k+1)$, assuming $\underline{q}(k)$ evolves according to Algorithm 1, is defined as

$$
\begin{aligned}
&E[q_1(k+1)|q_1(k)] = \\
&q_1(k)\big[c_1 q_1(k) + d_1(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k) + a\big(1-p_0^2(k)\big)\}\{p_0^3(k) + a\big(1-p_0^3(k)\big)\}\{p_0^4(k) + a\big(1-p_0^4(k)\big)\}\big] + \\
&q_2(k)\big[c_2 q_1(k) + d_2(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k)(1-a)\}\{p_0^3(k) + a\big(1-p_0^3(k)\big)\}\{p_0^4(k) + a\big(1-p_0^4(k)\big)\}\big] + \\
&q_3(k)\big[c_3 q_1(k) + d_3(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k) + a\big(1-p_0^2(k)\big)\}\{p_0^3(k)(1-a)\}\{p_0^4(k) + a\big(1-p_0^4(k)\big)\}\big] + \\
&q_4(k)\big[c_4 q_1(k) + d_4(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k) + a\big(1-p_0^2(k)\big)\}\{p_0^3(k) + a\big(1-p_0^3(k)\big)\}\{p_0^4(k)(1-a)\}\big] + \\
&q_5(k)\big[c_5 q_1(k) + d_5(k)\{p_1^1(k)(1-a)\}\{p_0^2(k)(1-a)\}\{p_0^3(k) + a\big(1-p_0^3(k)\big)\}\{p_0^4(k)(1-a)\}\big] + \\
&q_6(k)\big[c_6 q_1(k) + d_6(k)\{p_1^1(k)(1-a)\}\{p_0^2(k) + a\big(1-p_0^2(k)\big)\}\{p_0^3(k)(1-a)\}\{p_0^4(k)(1-a)\}\big] + \\
&q_7(k)\big[c_7 q_1(k) + d_7(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k)(1-a)\}\{p_0^3(k)(1-a)\}\{p_0^4(k) + a\big(1-p_0^4(k)\big)\}\big] + \\
&q_8(k)\big[c_8 q_1(k) + d_8(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k)(1-a)\}\{p_0^3(k) + a\big(1-p_0^3(k)\big)\}\{p_0^4(k)(1-a)\}\big] + \\
&q_9(k)\big[c_9 q_1(k) + d_9(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k) + a\big(1-p_0^2(k)\big)\}\{p_0^3(k)(1-a)\}\{p_0^4(k)(1-a)\}\big] + \\
&q_{10}(k)[c_{10} q_1(k) + d_{10}(k)\{p_1^1(k)(1-a)\}\{p_0^2(k)(1-a)\}\{p_0^3(k)(1-a)\}\{p_0^4(k)(1-a)\}] + \\
&q_{11}(k)\big[c_{11} q_1(k) + d_{11}(k)\{p_1^1(k) + a\big(1-p_1^1(k)\big)\}\{p_0^2(k)(1-a)\}\{p_0^3(k)(1-a)\}\{p_0^4(k)(1-a)\}\big]
\end{aligned}
$$

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

$$E[q_1(k+1)|q(k)] = \sum_{j=1}^{11} q_j(k)\left[c_j(k)q_1(k) + d_j(k)\left(\prod_{v_m \notin \omega_1} p_0^m(k) \prod_{v_m \in \omega_1} p_1^m(k)\right)\right]$$

Where

$$p_0^m(k+1) = \begin{cases} p_0^m(k) + a\big(1-p_0^m(k)\big) & ;\ v_m \notin \omega_j \\ p_0^m(k)(1-a) & ;\ v_m \in \omega_j \end{cases}$$

$$p_1^m(k+1) = \begin{cases} p_1^m(k) + a\big(1-p_1^m(k)\big) & ;\ v_m \in \omega_j \\ p_1^m(k)(1-a) & ;\ v_m \notin \omega_j \end{cases}$$

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

$$E[q_1(k+1)|q(k)] = \sum_{j=1}^{11} q_j(k)\left[c_j(k)q_1(k) + d_j(k)\left(\prod_{v_m \notin \omega_1} (1-p_1^m(k)) \prod_{v_m \in \omega_1} p_1^m(k)\right)\right]$$

where

$$p_1^m(k+1) = \begin{cases} p_1^m(k) + a\big(1-p_1^m(k)\big) & ;\ v_m \in \omega_j \\ p_1^m(k)(1-a) & ;\ v_m \notin \omega_j \end{cases}$$

and hence the proof of the lemma. ■

Lemma 3 Let$q_i(k)$denotes the probability of formation of the minimum WCDS 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

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

where

$$p_1^m(k+1) = \begin{cases} p_1^m(k) + a\left(1 - p_1^m(k)\right) & ; \ v_m \in \omega_j \\ p_1^m(k)(1-a) & ; \ v_m \notin \omega_j \end{cases}$$

For notational convenience, let

$$\prod_{\substack{\omega_i \\ n \in \{0,1\}}} \pi_n^m(k) = \prod_{v_m \notin \omega_i} (1 - p_1^m(k)) \prod_{v_m \in \omega_i} p_1^m(k)$$

Therefore, we have

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

Where

$$\pi_n^m(k+1) = \begin{cases} p_n^m(k) + a\left(1 - p_n^m(k)\right) & ; \ (v_m \notin \omega_j, n = 0) \ or \ (v_m \in \omega_j, n = 1) \\ p_n^m(k)(1-a) & ; \ (v_m \notin \omega_j, n = 1) \ or \ (v_m \in \omega_j, n = 0) \end{cases}$$

Where $\pi_n^m(k)$ is the probability with which vertex v_m, at stage k, declares itself as a dominator vertex, when $n = 1$, and as a dominatee vertex otherwise.

Since the probability that the WCDS is constructed, rewarded or penalized, is defined as the product of the probability of declaring the vertices in the constructed WCDS as dominators or dominatees as defined in equation (10), we have

$$\Delta q_i(k) = \sum_{j=1}^{r} \prod_{\substack{\omega_j \\ n \in \{0,1\}}} p_n^m(k) \left[\prod_{\substack{\omega_j \\ n \in \{0,1\}}} c_n^m(k) \prod_{\substack{\omega_i \\ n \in \{0,1\}}} p_n^m(k) + \prod_{\substack{\omega_j \\ n \in \{0,1\}}} d_n^m(k) \prod_{\substack{\omega_i \\ n \in \{0,1\}}} \pi_n^m(k) \right] - \prod_{\substack{\omega_i \\ n \in \{0,1\}}} p_n^m(k)$$

where $\pi_n^m(k)$ is defined as given in equation (11), $c_n^m(k)$ is the probability of penalizing the action chosen by vertex v_m at stage k, and $d_n^m(k) = 1 - c_n^m(k)$.

$$\Delta q_i(k) = \prod_{\substack{\omega_i \\ n \in \{0,1\}}} E[p_n^m(k+1)|p^m(k)] - \prod_{\substack{\omega_i \\ n \in \{0,1\}}} p_n^m(k)$$

The above quality can be rewritten as

$$\Delta q_i(k) \geq \prod_{\substack{\omega_i \\ n\in\{0,1\}}} E[p_n^m(k+1)|p^m(k)] - p_n^m(k) = \prod_{\substack{\omega_i \\ n\in\{0,1\}}} \Delta p_n^m(k) \tag{12}$$

where

$$\Delta p_n^m(k)_{n\in\{0,1\}} = a p_n^m(k) \sum_{s\neq n} p_s^m(k)\,[c_s^m(k) - c_n^m(k)]$$

and $q_i(k) \in (0,1)$ (for all $\underline{q} \in S_r^o$). S_r^o is defined as the interior of S_r, where $S_r = \{\underline{q}(k) | 1 \leq q_i(k) \leq 1, \sum_{i=1}^r q_i(k) = 1\}$. Hence, $p_n^m(k) \in (0,1)$ for all m, n. Since action α_n^m is the action with the minimum penalty probability which can be selected by automaton A_m, it is shown that $c_s^{m^*} - c_n^{m^*} > 0$, for all $s \neq n$, where $c_s^{m^*}$ is the final value to which the penalty probability $c_s^m(k)$ is converged. It follows from lemma 1 that for large values of k, $c_s^m(k) - c_n^m(k) > 0$. Therefore, we conclude that for large values of k, the right hand side of the equation above consists of the nonnegative quantities, and so we have

$$\prod_{\substack{\omega_i \\ n\in\{0,1\}}} a p_n^m(k) \sum_{s\neq n} p_s^m(k)\,[c_s^m(k) - c_n^m(k)] \geq 0$$

and from equation (12), we have

$$\Delta q_i(k) \geq \prod_{\substack{\omega_i \\ n\in\{0,1\}}} a p_n^m(k) \sum_{s\neq n} p_s^m(k)\,[c_s^m(k) - c_n^m(k)]$$

which completes the proof of this lemma. ■

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

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

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

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

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 [15] that operator U is linear and as the expectation of a nonnegative function remains nonnegative, operator U preserves the non-

negative functions. In other word, $U\psi(q) \geq 0$ for all $q \in S_r$, if $\psi(q) \geq 0$. If the operator U is repeatedly applied n times (for all $n > 1$), we have

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

A function $\psi(q)$ is called super regular (sub regular) if and only if $\psi(q) \geq U\,\psi(q)$ $(\psi(q) \leq U\psi(q))$, for all $q \in S_r$. It has also been shown in [15] that $\Gamma_i(q)$ is the only continuous solution of $U\Gamma_i(q) = \Gamma_i(q)$, with the following boundary conditions.

$$\Gamma_i(e_i) = 1$$
$$\Gamma_i\big(e_j\big) = 0 \quad ; \quad j \neq i \tag{14}$$

Define

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

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 $\psi_i\big(e_j\big) = 1$ for $j \neq i$, then
$\psi_i(q) \geq \Gamma_i(q)$
for all $q \in S_r$. If $\psi_i(.) \in C(S_r)$ is sub regular with the same boundary conditions, then
$\psi_i(q) \leq \Gamma_i(q)$ (15)
for all $q \in S_r$.

Proof Theorem 2 has been proved in [15]. ■

In what follows, we show that $\phi_i[x,q]$ is a sub regular function, thus $\phi_i[x,q]$ 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 U given in equation (13), and equation (11) we have

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

and

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

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 WCDS 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 WCDS ω_i.

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

and

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

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

$$\rho_j^i = \begin{cases} \prod_{\substack{(v_m \notin \omega_j, n=0) \\ or \\ (v_m \in \omega_j, n=1)}} \frac{p_n^m + a(1-p_n^m)}{p_n^m (1-a)} & ; \quad i \neq j \\ 1 \quad ; & (i=j)\ or\ (\omega_j \cap \omega_j = \emptyset) \end{cases}$$

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

$\theta_i[x,q]$ is super regular if

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

and

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

If $\theta_i[x,q]$ is super regular, so we have

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

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

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

and

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

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

where $G_i[x,q]$ is defined as

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

Therefore, $\theta_i[x,q]$ is super regular if

$$G_i[x,q] \ge 0 \tag{17}$$

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

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

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

$$\sum_{j\neq i} q_j \cdot \min_{j\neq i}\frac{d_j^*}{d_i^*} \le \frac{1}{1-q_i}\sum_{j\neq i} q_j \frac{d_j^*}{d_i^*} \le \sum_{j\neq i} q_j \cdot \max_{j\neq i}\frac{d_j^*}{d_i^*}$$

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

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

From equation (18), it follows that $\theta_i[x,q]$ is super regular if we have

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

For further simplification, let employ logarithms. Let

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

It has also been shown in [19] that

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

where

$$H'(u) = \frac{dH(u)}{du}$$

and

$$H(u) = \ln V(u)$$

We have

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

and

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

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

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

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

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

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 $0 < a \le a^*$, we have
$1 - \varepsilon \le \phi_i[x,q] \le \Gamma_i(q) \le 1$

Thus we conclude that the probability with which the Algorithm 1 converges to (or constructs) the WCDS 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 WCDS ω_i, at stage k, and $1-\varepsilon$ be the probability with which Algorithm 1 converges to the WCDS 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 $a \in (\varepsilon, q)$ so that,

$$\frac{xa}{e^{xa}-1} = \max_{j \ne i} \frac{d_j^*}{d_i^*}$$

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

Proof It has been proved in [15, 19] that there always exists a $x > 0$ under which the equation (19) is satisfied, if $\frac{d_j}{d_i} < 1$, for all $j \ne i$. Hence, it is concluded that

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

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

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

It can be shown that [19, 39] for every error parameter $\varepsilon \in (0,1)$ there exists a value of x under which the equation (19) is satisfied, and we have

$$\frac{x^* a}{e^{x^* a}-1} = \max_{j \ne i} \frac{d_j^*}{d_i^*}$$

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 WCDS with the minimum expected weight is greater than $1-\varepsilon$ and hence the proof of the theorem. ■

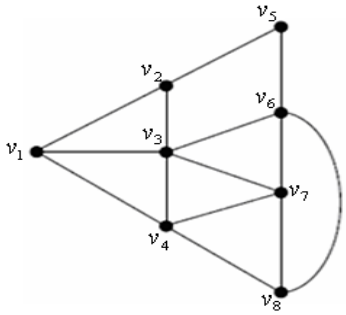
5. Experimental Results

To study the performance of the proposed algorithms, we have conducted two groups of simulation. The first group is concerned with investigating the efficiency of the proposed algorithms on three

stochastic graphs borrowed from [40] and shown in figures 8, 9, and 10, and the second group of the simulation evaluates the scalability of the proposed algorithms on dense stochastic graphs. In all the experiments presented in this paper, the number of samples taken by the proposed algorithms from the stochastic graph to construct the WCDS with the minimum expected weight is compared with that of the SSM quoted in appendix A. The reinforcement scheme used for updating the action probability vectors is L_{R-I}, and each algorithm is terminated when the probability of the constructed WCDS (*PWCDS*) is 0.95 or greater, or the number of constructed WCDSs exceeds a pre-defined threshold.

In the first group of the simulations, the proposed algorithms are tested on graphs 1, 2, and 3, and the results are summarized in Tables 2, 4, and 6 for the different values of learning rate a which varies from 0.004 to 0.100. Then, the obtained results in terms of the number of samples taken from the graph are compared with those of the SSM given in Tables 1, 3, and 5. The aim of the SSM used in our experiments is to obtain the minimum number of samples needs to be taken from the stochastic graph such that the average weight of the samples taken from the optimal WCDS be 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 Figure 8 has 8 vertices, 12 edges and its minimal WCDS is $\delta_1^* = \{v_1, v_6\}$, graph2 which is shown in Figure 9 has 9 vertices, 15 edges and its minimal WCDS is $\delta_2^* = \{v_1, v_6\}$, and graph3 which is shown in Figure 10 has 10 vertices, 21 edges and its minimal WCDS is $\delta_3^* = \{v_1, v_7\}$. The probability distributions of the vertex weight of these three stochastic graphs are given in Tables shown in Figures 8, 9, and 10. The simulation results given in Tables 2, 4, and 6 are averaged over 1000 runs and each of these Tables includes the information about the total number of samples taken from all vertices of the graph (TS), the total number of the samples taken from the vertices of the WCDS with the minimum expected weight (WCDS) and the percentage of the converged runs (the percentage of runs converged to the minimum WCDS) (PC).

According to the SSM described in appendix A, to obtain a confidence level $1 - \varepsilon$ for the WCDS, we need to build a confidence with level $1 - \varepsilon_i$ for each vertex v_i such that $\sum_{i=1}^{k} \varepsilon_i = \varepsilon$. We assume that the vertices of the stochastic graph all have the same confidence level $1 - \varepsilon_0$. Therefore, selecting $\varepsilon_0 = \frac{\varepsilon}{k}$, where k denotes the cardinality of the WCDS, guarantees a confidence level $1 - \varepsilon$ for the WCDS. For instance, when a confidence level 80% is desired for the WCDS of graph 1, we need to build a confidence with level 90% for each of the 8 vertices [43]. The results of the standard sampling method for graphs 1, 2, and 3 to obtain a confidence level $1 - \varepsilon$ for the WCDS, as ε varies from 0.01 to 0.5 have been shown in Tables 1, 2, and 3, respectively.



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}

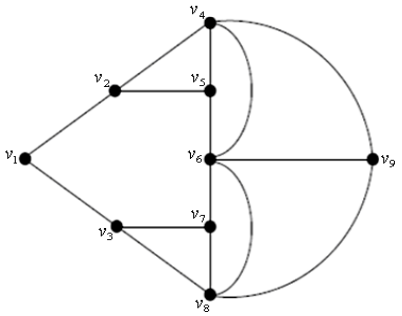
Figure 8. Stochastic graph 1 and its probability distribution function

Table 1. The total number of samples taken from graph 1 in SSM

Vertex	Confidence Level for WCDS										
	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. The average number of samples taken from the graph and the minimum WCDS, and the percentage of the converged runs for graph 1

Learning	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5		
rate	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC
0.004	4909.7	1675.4	100.0	2719.3	1994.3	100.0	13732.3	7163.4	100.0	2433.1	1035.2	100.0	3750.8	1454.8	100.0
0.005	3945.8	1347.0	100.0	2267.1	1661.6	100.0	11463.9	6102.7	100.0	1953.1	831.0	100.0	3079.0	1450.1	100.0
0.006	3332.0	1129.5	100.0	1947.5	1424.4	100.0	9468.3	5506.4	100.0	1647.9	692.8	100.0	2499.4	1091.2	99.0
0.007	2844.6	963.5	98.5	1707.1	1248.6	100.0	4762.8	1305.1	100.0	1412.8	594.4	100.0	2277.4	1060.1	97.0
0.008	2495.3	843.6	98.2	1517.1	1110.5	100.0	4499.3	1359.0	100.0	1237.0	518.9	100.0	2089.7	1058.3	99.0
0.009	2222.7	750.0	97.8	1369.3	1000.8	100.0	4074.5	1377.3	97.0	1102.0	461.8	100.0	2054.7	1129.5	98.0
0.01	2021.3	675.8	96.7	686.6	499.0	100.0	3281.9	2030.9	60.4	1003.7	418.1	99.6	1682.5	840.3	97.2
0.02	1015.8	331.3	87.5	456.6	331.6	100.0	2127.5	1356.5	47.4	553.2	224.7	96.9	1000.1	589.3	93.8
0.03	673.5	213.3	81.6	346.4	250.5	100.0	1283.5	730.9	32.0	395.0	173.6	91.9	588.6	326.0	90.8
0.04	496.8	155.4	73.0	276.2	197.0	100.0	1341.8	800.0	33.7	292.3	131.8	90.4	431.8	250.3	82.4
0.05	386.8	119.1	70.1	228.1	162.8	100.0	943.5	532.4	22.1	229.2	104.8	87.9	279.5	141.1	77.2
0.06	314.3	95.5	59.4	198.4	139.2	100.0	677.4	368.4	14.6	182.8	82.9	82.1	234.6	117.0	76.4
0.07	268.9	83.3	54.3	196.0	138.7	100.0	523.9	269.2	13.1	152.1	70.0	79.8	204.2	109.9	75.2
0.08	233.2	72.1	52.8	170.8	120.2	99.0	423.3	218.3	8.5	129.5	60.9	79.4	169.8	92.7	70.8
0.09	199.1	61.1	46.7	154.5	108.3	97.8	352.4	168.6	6.3	116.7	54.8	71.2	133.7	69.7	70.4
0.1	181.2	56.0	43.6	138.3	96.0	97.4	300.4	151.1	4.8	101.5	46.9	68.0	124.7	70.1	70.0



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}

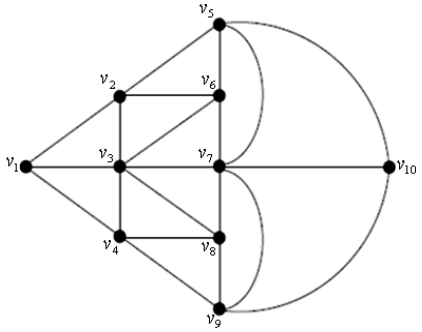
Figure 9. Stochastic graph 2 and its probability distribution function

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

Vertex	Confidence Level for WCDS										
	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
Total	3413	3437	3418	3492	3464	3632	3743	3627	3773	4027	4061

Table 4. The average number of samples taken from the graph and the minimum WCDS, and the percentage of the converged runs for graph 2

Learning rate	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5		
	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC
0.004	4713.0	1259.4	100.0	4569.5	2783.0	100.0	6513.1	1039.1	100.0	3791.4	1062.8	100.0	2989.5	981.4	100.0
0.005	3794.3	1001.7	100.0	3699.5	2231.2	100.0	5585.8	839.4	100.0	3080.7	869.2	100.0	2721.2	1106.1	100.0
0.006	3183.4	836.9	99.9	3043.8	1861.7	100.0	4759.9	702.6	100.0	2536.6	716.2	100.0	2146.7	702.1	100.0
0.007	2741.1	715.3	99.5	2653.2	1592.5	100.0	4165.7	601.4	100.0	2181.6	609.4	100.0	1923.4	644.6	99.6
0.008	2412.5	622.9	99.4	2322.0	1395.4	100.0	3723.0	527.9	100.0	1964.5	535.7	98.0	1733.8	619.4	100.0
0.009	2151.5	553.4	99.3	2040.6	1244.2	100.0	3389.0	468.9	100.0	1744.1	558.6	96.0	1632.6	632.7	99.0
0.01	1928.6	498.0	99.1	1840.2	1113.5	99.1	3048.8	421.6	99.6	1725.3	570.6	94.0	1433.7	487.5	99.0
0.02	983.4	241.6	94.8	926.1	557.5	96.0	2241.9	825.6	78.6	1619.8	499.5	93.0	836.3	377.0	91.0
0.03	660.2	159.4	87.7	618.6	371.4	92.9	2193.8	1370.7	47.6	1139.2	663.5	69.0	546.2	233.8	89.4
0.04	488.5	116.2	83.3	460.9	278.8	90.0	1640.1	1074.1	28.6	806.9	492.7	64.0	401.9	180.7	87.0
0.05	392.9	94.6	75.4	371.3	222.7	86.6	1074.0	633.4	24.8	371.2	206.6	56.5	297.6	128.2	83.2
0.06	323.9	76.8	66.6	303.2	184.0	84.9	850.3	516.4	12.8	354.9	186.6	56.0	220.2	91.0	75.4
0.07	268.2	65.7	66.1	261.7	158.6	85.0	590.0	315.4	13.8	211.7	107.8	53.9	186.3	78.6	78.8
0.08	236.3	61.0	51.6	224.0	136.9	82.8	478.7	246.6	6.2	175.8	88.2	55.0	157.9	70.8	75.6
0.09	212.9	53.1	52.5	199.1	121.4	82.3	419.0	220.9	5.8	148.9	75.1	51.2	137.6	60.2	74.2
0.1	190.3	53.6	45.5	176.9	109.3	80.6	353.3	190.5	4.8	129.6	64.2	48.3	132.1	57.7	73.4



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}

Figure 10. Stochastic graph 3 and its probability distribution function

Table 5. The total number of samples taken from graph 3 in SSM

Vertex	Confidence Level for WCDS										
	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

Table 6. The average number of samples taken from the graph and the minimum WCDS, and the percentage of the converged runs for graph 3

Learning	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4			Algorithm 5		
rate	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC	TS	WCDS	PC
0.004	5561.5	1543.4	100.0	4301.1	2699.1	100.0	15981.1	7422.8	100.0	2826.1	1056.4	100.0	5098.2	2892.7	100.0
0.005	4494.1	1245.4	97.2	3435.9	2158.4	100.0	10499.5	3579.0	100.0	2284.6	845.1	100.0	4098.9	2263.0	100.0
0.006	3779.9	1045.6	96.6	2880.9	1801.8	100.0	7549.3	1922.4	100.0	1913.1	702.2	100.0	2989.1	1207.4	100.0
0.007	3262.4	902.4	93.9	2468.4	1542.9	100.0	6697.9	1660.3	100.0	1656.4	602.4	100.0	2659.4	1131.2	100.0
0.008	2843.0	777.5	93.0	2164.2	1352.2	100.0	5919.9	1561.6	100.0	1451.2	529.3	100.0	2338.4	1046.5	98.6
0.009	2551.7	699.4	91.0	1927.9	1202.0	100.0	5100.9	963.7	100.0	1300.6	472.1	100.0	2245.6	1136.3	97.8
0.01	2309.9	626.6	90.4	1734.0	1080.9	100.0	4781.7	1243.3	96.0	1187.3	429.8	99.8	2034.1	1011.9	97.6
0.02	1131.0	296.4	82.1	871.6	539.5	99.5	3918.5	2294.2	36.8	652.4	244.3	97.0	1398.3	943.1	91.0
0.03	750.1	187.2	71.7	587.3	360.4	98.4	3096.1	2135.7	15.8	439.3	158.5	92.9	775.3	483.3	83.4
0.04	551.7	129.8	70.8	440.3	270.4	96.7	1948.3	1304.2	11.0	354.8	140.5	84.9	522.3	307.4	83.0
0.05	436.9	104.8	59.4	352.9	216.5	96.3	1298.7	813.6	8.0	275.2	109.7	78.9	266.9	139.0	81.0
0.06	353.3	87.1	54.0	289.5	178.0	94.6	893.6	514.8	7.1	234.6	96.6	73.1	236.0	126.1	80.0
0.07	303.4	65.2	48.7	251.7	153.3	90.9	675.7	369.9	6.7	193.7	79.7	68.5	206.4	99.6	74.8
0.08	256.5	63.5	41.4	221.2	135.2	89.0	517.3	262.6	5.8	165.8	68.0	63.9	176.3	87.6	74.8
0.09	223.6	54.2	39.7	193.0	119.7	87.4	412.5	194.1	5.0	142.1	60.0	61.0	160.1	78.3	72.0
0.1	197.9	50.5	35.0	174.8	107.1	84.4	346.9	166.5	4.6	125.0	55.8	56.6	142.1	69.4	71.0

From the results of the simulations, the following points can be made:

The experiments have shown 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 54.3%, when $a = 0.07$, whereas it is 98.5% when $a = 0.007$. (see Table 2)

The simulation results show that the total number of samples taken by Algorithm 2 from the vertices of the graph is less than those of Algorithm 1, but the percentage of the runs converged to the optimal WCDS is higher. The reason for this reduction in number of samples taken by Algorithm 2 from the graph is the fact that, when a vertex declares itself as a dominator and its weight is sampled, unlike Algorithm 1, Algorithm 2 does not need to sample the weight of its neighbors again. This sampling method results in decreasing unnecessary samples and hence decreasing the running time of Algorithm 2. The results show that for the same convergence rate, Algorithm 2 has lower sampling rate comparing Algorithm 1.

Experiments show that Algorithm 3 has a lower convergence rate comparing Algorithm 2. This may be due to the fact that the action-set of each learning automaton in Algorithm 3 is significantly larger than that of in Algorithm 2, specifically for the dense graphs. That is, in Algorithm 3 as stated in Section 3, each

learning automaton has many choices (the number of actions available for each automaton is equal to the number of its neighbors plus one), while in Algorithm 2 each learning automaton has at most two possible actions. This results in prolonging the convergence of each learning automaton to its optimal action, and so postpones the convergence of the proposed algorithm to the optimal WCDS.

Comparing the results given in Tables 2, 4, and 6 for algorithms 1 and 3, we observe that the total number of samples taken by Algorithm 3 is not much more than that of Algorithm 1, while it is expected to be due to the larger number of actions in Algorithm 3. The reason for this reduction in number of samples taken by Algorithm 3 can be the fact that Algorithm 3 guarantees formation of a WCDS at each stage, while in a number of stages Algorithm 1 fails in constructing a WCDS for the stochastic graph, which results in increasing the number of samples needs to be taken by the algorithm.

The simulation results show that Algorithm 4 greatly improves the convergence rate. The reason for higher convergence rate for this algorithm is the fact that it employs the variable action-set learning automata to construct the WCDSs which are more likely to be the WCDS with the minimum expected weight, and so increases the convergence rate to the optimal WCDS. The results show that Algorithm 4 also considerably outperforms algorithms 1, 2, 3, and 5 in terms of the sampling rate.

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

Comparing the results given in Tables 1, 3, and 5 with Tables 2, 4, and 6, we find that the number of samples needs to be taken by the proposed algorithms, when the algorithms converge to the minimum WCDS with probability $1 - \varepsilon$, is considerably less than that of the SSM. This study is the main result of the paper that was theoretically proved in the previous section and will be considered in more details at the end of this section.

Another property we want to illustrate is any time behavior of the proposed algorithms. We show this property by plotting the development of the probability of choosing the minimum WCDS in time. Figures 11, 12 and 13 show a rapid progress in the beginning and flattening out later on. This is typical for many algorithms that work by iterative improvements on the initial solution. Based on the any time curve depicted in Figures 11, 12 and 13 we can make some general observation concerning termination condition for the proposed algorithms. We can divide the run time into two equally long sections, the first and the second half. As Figures 11, 12 and 13 indicate, the progress in terms of increments in probability of choosing the minimum WCDS is significantly greater than the achievements in the second half. This provides a general suggestion that it might not be worthwhile to allow very long runs; because of the any time behavior of the proposed algorithms efforts spent after a certain time may not result in better solution quality. A given point in any time curve gives us the average number of iterations required to find the minimum WCDS with a certain probability. This is one advantage of the proposed algorithm over non-iterative algorithms. Probability of the minimum WCDS (PWCDS) at each instant of the proposed algorithms is defined as the product of the probability of choosing the vertices of the constructed WCDS. The plots of the average probability of the minimum WCDS over the converged runs out of 100 runs for both graphs and different algorithms are given in Figures 11, 12 and 13. Comparing the any time behavior of the proposed algorithms, we observe that Algorithm 1 performs best and Algorithm 3 performs worst among the algorithms presented in this paper.

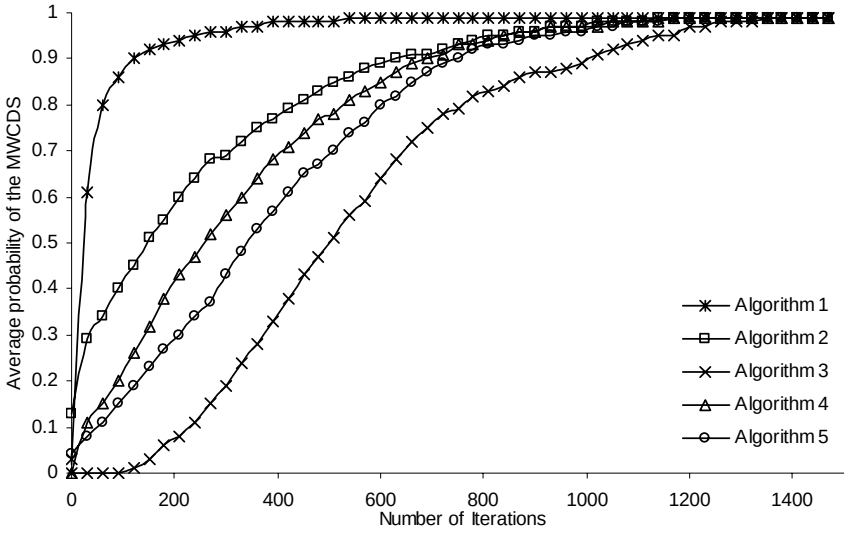


Figure 11. The probability of the minimum WCDS for graph 1

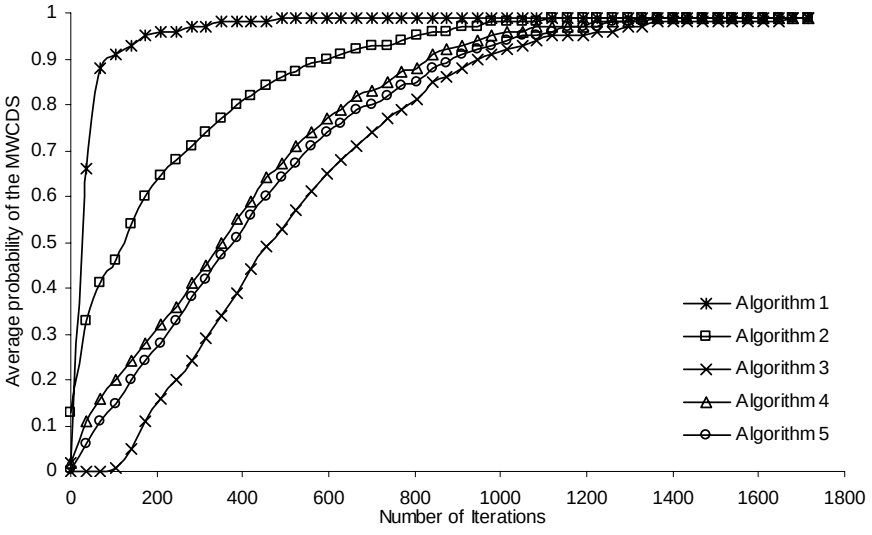


Figure 12. The probability of the minimum WCDS for graph 2

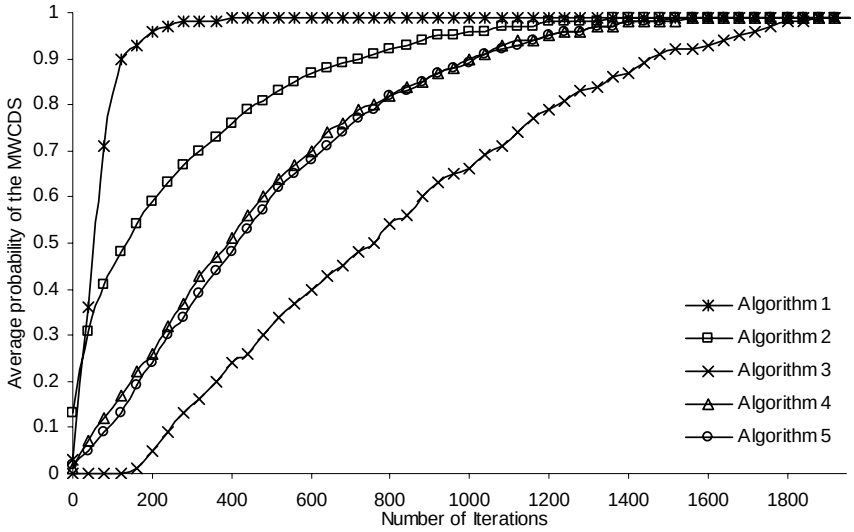


Figure 13. The probability of the minimum WCDS for graph 3

To evaluate the scalability of the proposed algorithms for large graphs, in the second group of the simulations, the proposed algorithms are tested on the dense stochastic graphs. The stochastic graphs used in these experiments are generated based on the method proposed in [44] to create the weighted geometric graphs. The weight of each vertex v_i $\left(\text{for all } 1 \leq i \leq n\right)$ is assumed to be an exponential random variable with parameter $\lambda = \frac{i}{n}$, where n is the number of vertices in the graph. It is assumed that the parameters of the underlying probability distribution of the vertex weight are unknown. In such geometric graphs, the vertices are randomly and uniformly distributed in a square area, and an edge connects two given vertices, if they are close enough (if the distance between to vertices is less than r).

In these experiments, it is assumed that a given number of vertices are randomly and uniformly distributed in a square area of size 100×100 units, and each two given vertices are connected together if the distance between them is at most 20. The total number of samples taken from the stochastic graphs by the proposed algorithms is obtained when the number of vertices ranges from 20 to 150 with increment step of 10 and the percentage of the converged runs is 98 percent. All the simulation results given in Table 7 are obtained by running the proposed algorithm on 100 stochastic graphs and averaged over these runs. Table 7 includes the information about the total number of the samples taken from all vertices of the graph (TS) and the total number of the samples taken from the vertices of the minimum WCDS (WCDS) for each of the proposed algorithms as well as the total number of samples taken by the SSM when the confidence level for the WCDS is 98 percent.

The simulation results of the proposed algorithms given in Table 7 show that Algorithm 2 performs best and Algorithm 3 performs worst among the proposed algorithms. The obtained results also show that the number of samples taken by each of the five proposed algorithms increases as the number of vertices increases. Comparing the total number of samples taken by the proposed algorithms with that of the SSM, we observe that the proposed algorithms significantly outperform the SSM in terms of the sampling rate, especially for the dense graphs.

Table 7. The sampling rate of the proposed algorithms against the SSM

Number of Vertices	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4		Algorithm 5		SSM
	TS	WCDS	TS	WCDS	TS	WCDS	TS	WCDS	TS	WCDS	
20	4796.5	1014.0	2594.7	835.9	9216.5	5606.0	3248.2	1763.4	6699.2	3300.5	7663.0
30	5940.0	1052.9	2916.4	850.4	7936.5	5445.0	3080.5	1119.0	7836.8	3456.7	13186.0
40	6975.0	1002.0	3342.2	890.5	9657.0	6513.5	4070.5	2066.5	8304.6	3006.2	18949.0
50	7874.5	1042.2	3510.3	879.4	10671.0	6572.4	4583.3	2690.0	8985.4	3046.3	22134.0
60	8733.2	1098.9	4034.0	868.9	11328.4	7510.0	5259.0	2395.0	9023.9	3721.2	26894.0
70	9045.3	983.0	4084.6	894.5	15076.2	9269.6	5445.0	2825.0	9589.0	3960.5	30335.0
80	9559.2	983.2	4216.2	865.8	15640.9	9291.7	5670.5	3224.5	10746.5	4172.6	33561.0
90	10520.3	1213.8	4331.1	914.6	16652.1	9625.4	5930.4	3311.3	12184.5	4024.9	39505.0
100	11428.0	1117.1	4552.1	894.3	17186.5	10734.7	6040.5	3519.5	12322.0	4847.0	43686.0
110	11494.1	1112.9	5078.6	921.2	19750.0	13987.0	6570.3	3210.2	14309.8	3598.2	49083.0
120	12745.7	1215.6	4953.0	902.6	20579.0	10028.0	6700.1	3636.0	19593.8	7558.8	50969.0
130	11876.0	1071.7	5122.8	893.4	24869.5	13422.5	6642.2	3515.5	19311.5	4183.5	58787.0
140	13091.4	1129.2	5222.8	910.4	27128.0	17057.0	7010.5	3710.0	21345.4	11618.5	64476.0
150	13146.1	967.2	5276.4	900.8	31656.0	21239.0	7230.0	3685.3	23916.7	11012.2	66469.0

The remainder of the simulation experiments compares the first proposed algorithm (*Algorithm 1*) with the standard sampling method in terms of the number of samplings. Let q_i denotes the initial probability of constructing the minimum WCDS ω_i, and $1-\varepsilon$ denotes the probability with which Algorithm 1 converges to the WCDS ω_i and $\underline{q}(\mathrm{k})$ 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 the equation below

$$\frac{1-e^{-xq_i}}{1-\varepsilon} = 1-e^{-x} \tag{20}$$

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

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

The equation above can be solved by the numerical methods. Learning rate a is then calculated for the given parameter x by solving equation (21)

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

The number of samples needs to be taken by Algorithm 1 is then determined by running the algorithm for the learning rate obtained from the equation above. The minimum number of samples taken by the standard sampling method, satisfying $prob[|\overline{x}_n - \mu| < \delta] \geq 1-\varepsilon$, has also been given in Tables 1, 3, and 5, 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 given in Table 8.

Table 8. A comparison of sampling rate of Algorithm 1 and SSM for graph 1

Convergence Rate	Learning rate	Sampling Rate	
		Algorithm 1	SSM
0.50	0.0880	255.02	3171
0.60	0.0690	315.79	3187
0.70	0.0434	550.29	3118
0.75	0.0336	715.22	3149
0.80	0.0281	862.60	3152
0.85	0.0211	1154.22	3261
0.90	0.0122	1955.25	3404
0.95	0.0088	2752.33	3445
0.99	0.0079	3049.92	3628

6. Conclusion

In this paper, several learning automata-based algorithms were proposed to solve the minimum WCDS problem in a stochastic graph when the probability distribution function of the weight of the vertices is unknown. To evaluate the proposed algorithms, at first, the number of samples needs to be taken from the graph was obtained by the standard sampling method when the approximated value of the vertex weight satisfies a certain confidence constraint. Then, the number of samples taken by the proposed algorithm, where the probability with which the algorithm converges to the WCDS with the minimum expected weight satisfies the given constraint, were calculated and shown to be much less than that of the standard sampling method. As the algorithms proceed, taking advantage of learning automata the process of sampling from the graph is concentrated on the vertices by which the minimum WCDS is constructed. Such an intelligent sampling process meaningfully reduces the number of samples needs to be taken from the graph. For the first proposed algorithm, it was shown that by a proper choice of the parameters of the learning algorithm, the probability with which the given algorithm finds the optimal WCDS is close enough to unity. The simulation results showed the efficiency of the proposed algorithms in terms of the sampling rate.

Appendix A: Standard Sampling Method (SSM)

Theorem 4 To obtain a confidence level not smaller than $1-\varepsilon$ for the WCDS, 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 WCDS.

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

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

$$prob[|\overline{x}_N - \mu| < \delta] > 1-\varepsilon_i \tag{A.1}$$

for all $N > N_0$.

The problem is then to find a confidence region for each of the vertices under which a desired confidence level $1-\varepsilon$ is guaranteed for the WCDS. The confidence region for the WCDS is defined as the intersection $\bigcap_{i=1}^{k} C_i(\varepsilon_i)$ of the confidence regions for the vertices, where $C_i(\varepsilon_i) = 1-\varepsilon_i$ denotes the confidence region for vertex v_i and k denotes the cardinality of the WCDS. Using Booles-Bonferroni inequality [41], we have

$$\min_{1\le i\le k} 1-\varepsilon_i > prob\left[\mu_i \in \bigcap_{i=1}^{k} C_i(\varepsilon_i)\right] \ge 1-\sum_{i=1}^{k} prob[\mu_i \notin C_i(\varepsilon_i)] \tag{A.2}$$

and so

$$\min_{1\le i\le k} 1-\varepsilon_i > prob\left[\mu_i \in \bigcap_{i=1}^{k} C_i(\varepsilon_i)\right] \ge 1-\sum_{i=1}^{k} \varepsilon_i \tag{A.3}$$

Hence, the confidence level of the WCDS 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 WCDS. To achieve this, according to the Bonferroni Correction [42] it is sufficient to build a confidence with level $1-\varepsilon_i$ for each vertex v_i such that $\sum_{i=1}^{k}\varepsilon_i = \varepsilon$, and hence the proof of the theorem. ■

References

1. Y. P. Chen, A. L. Liestman, "Maintaining Weakly-Connected Dominating Sets for Clustering Ad Hoc Networks," *Ad Hoc Networks*, Vol. 3, pp. 629–642, 2005.

2. P. Gupta, P.R. Kumar, "The Capacity of Wireless Networks," *IEEE Transaction on Information Theory*, Vol. 46, No. 2, pp. 388–404, 2000.

3. B. Han, W. Jia, " Clustering Wireless Ad Hoc Networks with Weakly Connected Dominating Set," *Journal of Parallel and Distributed Computing*, Vol. 67, pp. 727 – 737, 2007.

4. R. Rajaraman, "Topology Control and Routing in Ad Hoc Networks: A Survey," *SIGACT News*, Vol. 33, No. 2, pp. 60–73, 2002.

5. B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit Disk Graphs," *Discrete Mathematics*, Vol. 86, pp. 165-177, 1990.

6. M.V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, D.J. Rosenkrantz, "Simple Heuristics for Unit Disk Graphs," *Networks* Vol. 25, pp. 59–68, 1995.

7. Y.Z. Chen, A.L. Listman, "Approximating Minimum Size Weakly Connected Dominating Sets for Clustering Mobile Ad hoc Networks," *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'2002)*, pp. 157–164, 2002.

8. Y.P. Chen, A.L. Liestman, "A Zonal Algorithm for Clustering Ad Hoc Networks," *International Journal of Foundations of Computer Science*, Vol. 14, No. 2, pp. 305–322, 2003.

9. 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*, Vol. 14, No. 2, pp. 287-303, 2003.

10. R.G. Gallager, P.A. Humblet and P.M. Spira, "A Distributed Algorithm for Minimum Weight Spanning Trees," *ACM Transaction on Programming Languages and Systems,* Vol. 5, pp. 66-77, 1983.

11. S. Guha and S. Khuller, "Approximation Algorithms for Connected Dominating Sets," *Algorithmica*, Vol. 20, No. 4, pp. 374-387, 1998.

12. O. Dousse, F. Baccelli, and P. Thiran, "Impact of Interferences on Connectivity in Ad hoc Networks," *IEEE/ACM Transactions on Networking*, Vol. 13, No. 2, pp. 425-436, 2005.

13. S. Basagni, M. Mastrogiovanni, C. Petrioli, "A Performance Comparison of Protocols for Clustering and Backbone Formation in Large Scale Ad Hoc Network," *Proceedings of the First IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS'2004)*, pp. 70–79, 2004.

14. B. Das, and V. Bharghavan, "Routing in ad-hoc networks using minimum connected dominating sets," *IEEE International Conference on Communications (ICC'97)*, 1997.

15. K. S. Narendra and K. S. Thathachar, "Learning Automata: An Introduction", New York, *Prentice-Hall*, 1989.

16. M. A. L. Thathachat, P. S. Sastry, "A Hierarchical System of Learning Automata That Can Learn the Globally Optimal Path," *Information Science*, Vol.42, pp.743-766, 1997.

17. M. A. L. Thathachar and B. R. Harita, "Learning Automata with Changing Number of Actions," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMG17, pp. 1095-1100, 1987.

18. M. A. L. Thathachat, V.V.Phansalkar, "Convergence of Teams and Hierarchies of Learning Automata in Connectionist Systems," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 24, pp. 1459-1469, 1995.

19. S. Lakshmivarahan and M. A. L. Thathachar, "Bounds on the Convergence Probabilities of Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics,* 1976, Vol. SMC-6, pp. 756-763, 1976.

20. K. S. Narendra, and M. A. L. Thathachar, "On the Behavior of a Learning Automaton in a Changing Environment with Application to Telephone Traffic Routing," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-l0, No. 5, pp. 262-269, 1980.

21. H. Beigy, M. R. Meybodi, "Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problems," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, Vol.14, pp. 591-615, 2006.

22. R. Ghosh, and S. Basagni, "Mitigating the Impact of Node Mobility on Ad Hoc Clustering," *Journal of Wireless Communications and Mobile Computing*,Vol. 8, pp. 295-308 , 2008.

23. J. Wu, B. Wu, and I. Stojmenovic, " Power-Aware Broadcasting And Activity scheduling in Ad Hoc Wireless Networks Using CDSs," *Journal of Wireless Communications and Mobile Computing*,Vol. 3, pp. 425-438 , 2003.

24. S. Basagni, D. Bruschi, I. Chlamtac, "A Mobility-Transparent Deterministic Broadcast Mechanism for Ad Hoc Network," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, pp-799-807, 1999.

25. I. Chlamtac, S. Kutten, "Tree-Based Broadcasting in Multihop Radio Networks," *IEEE Transactions on Computing*, Vol. 36, No. 10, pp. 1209-1223, 1987.

26. H. Lim, and C. Kim, "Flooding in Wireless Ad Hoc Networks," *Journal of Computer Communications*, Vol. 24, pp. 353-363, 2001.

27. X. Cheng, M. Ding, D. Hongwei, and X. Jia, " Virtual Backbone Construction in Multihop Ad Hoc Wireless Networks," *Journal of Wireless Communications and Mobile Computing*,Vol. 6, pp. 183-190, 2006.

28. F. D. Tolba, D. Magoni, and P. Lorenz, "A Stable Clustering Algorithm for Highly Mobile Ad Hoc Networks," *Second International Conference on Systems and Networks Communications (ICSNC 2007)*, 2007.

29. Y. X. Wang, and F. S. Bao "An Entropy-based Weighted Clustering Algorithm and Its Optimization for Ad hoc Networks," *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2007)*, 2007.

30. Z. E. Bazzal, M. Kadoch, B. L. Agba, F. Gagnon, and M. Bennani "A Flexible Weight Based Clustering Algorithm in Mobile Ad hoc Networks," *Proceedings of the International Conference on Systems and Networks Communication (ICSNC 2006)*, pp. 50-56, 2006.

31. T. N. Janakiraman, and J. L. Rani, "An Efficient K-Hop Weighted Domination Clustering Algorithm for Mobile Ad Hoc Networks Using Ranking," *International Conference on Computational Intelligence and Multimedia Applications*, pp. 322-326, 2007.

32. W. D. Yang, and G.Z. Zhang, "A Weight-Based Clustering Algorithm for mobile Ad Hoc network," *Proceedings of the Third International Conference on Wireless and Mobile Communications (ICWMC 2007)*, 2007.

33. W. Choi, and M. Woo, "A Distributed Weighted Clustering Algorithm for Mobile Ad Hoc Networks," *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, 2006.

34. R. Mellier, and J. F. Myoupo, "A Weighted Clustering Algorithm for Mobile Ad Hoc NetworksWith Non Unique Weights," *International Conference on Wireless and Mobile Communications (ICWMC 2006)*, 2006.

35. M. Gerla, J.T.-C. Tsai, "Multicluster, Mobile, Multimedia Radio Network," *Journal of Wireless Networks*, Vol. 1, No. 3, pp. 255–265, 1995.

36. J.E. Dunbar, J.W. Grossman, J.H. Hattingh, S.T. Hedetniemi, and A.A. McRae, "On Weakly Connected Domination in Graphs," *Discrete Mathematics*, Vol. 167–168, pp. 261–269, 1997.

37. I. Chlamtac, M. Conti, J. Liu, “Mobile Ad hoc Networking: Imperatives and Challenges,” *Journal of Ad Hoc Networks*, Vol. 1, pp. 13–64, 2003.

38. S. Basagni, M. Conti, S. Giordano and I. Stojmenovic, “Mobile Ad Hoc Networking,” *IEEE Press*, 2004.

39. S. Lakshmivarahan, and K. S. Narendra, “Learning Algorithms for Two Person Zero-Sum Stochastic Games with Incomplete Information: A Unified Approach,” *SIAM Journal of Control Optim.*, 1982, Vol. 20, pp. 541–552.

40. K. R. Hutson, and, D. R. Shier, "Minimum Spanning Trees in Networks with Varying Edge Weights," *Annals of Operations Research, Springer*, Vol. 146 pp. 3-18, 2006.

41. F. B. ALT, “Bonferroni Inequalities and Intervals,” *in Encyclopedia of Statistical Sciences*, Vol. 1, pp. 294–301, 1982.

42. 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, pp. 3–62, 1936.

43. S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," *Scandinavian Journal of Statistics*, Vol. 6, pp. 65–70, 1979.

44. Http://dimacs.rutgers.edu/Challenges