Learning Automata-Based Data Aggregation Tree Construction Framework for Cyber-Physical Systems

Jiankang Ren, Guowei Wu, Xiaoyan Su, Guanghui Cui, Feng Xia, Senior Member, IEEE, and Mohammad S. Obaidat, Fellow, IEEE

Abstract—A high degree of energy efficiency and real-time for data transmission is required in cyber-physical systems (CPSs). Data aggregation is an efficient technique to conserve energy by reducing the amount of transmission data. To optimize real-time communication under constraints of power consumption and data aggregation performance of each node in CPS, this paper presents a learning automata (LA)-based degree-bounded bottleneck data aggregation tree (DBBDAT) construction framework to minimize the maximum delay on data aggregation trees with bounded degree, which is an NP-hard problem. We model the network of CPS as a connected weighted and directed graph to form a network of LA. Degree-bounded data aggregation trees are constructed first by the action selection of each automaton. Then, the action vector of each automaton is updated by linear reward-inaction learning algorithm, and at last DBBDAT is constructed based on a threshold. Simulation results show that our approach significantly outperforms integer linear programming (ILP)-based method in terms of time complexity. Compared with ILP-based method, it can obtain an optimal solution or a suboptimal solution with guaranteed approximation ratios, and can control the trade-off between accuracy and cost by choosing appropriate learning rate and threshold. Its distributed implementation is simple and it can efficiently solve the problem for the sparse graph in practice.

Index Terms—Cyber-physical system (CPS), data aggregation tree, energy-efficient, learning automata (LA), real-time.

I. INTRODUCTION

Cyber-physical systems (CPSs) are engineered systems that are built from and depend on the tight integration of computational and physical components [1]. As a network of sensors, actuators, and embedded devices, CPS forms a part of safety-critical systems and infrastructure to sense, monitor, and control the physical world in real time to enhance the quality of life. For example, medical CPSs (MCPSs) bring significant impacts to the scope of medical services in areas such as rehabilitation, geriatric care, sports medicine, fall detection, and gait analysis [2]. To fully explore the potential of CPS, however, we need to overcome significant challenges [3].

Most of applications governed by CPS usually are geographically distributed, and their data collection is widely realized via wireless sensor networks (WSNs) with a routing tree structure: the observation at multiple sensors is sent to a controller or actuator (i.e., the sink node) through single hop or multiple hops. Wireless sensor nodes in CPS are equipped with only limited data processing capability, battery energy, and memory. In particular, sensor nodes are powered by batteries which cannot be replaced optionally due to the fact that they tend to be deployed in unattended environments. Therefore, it is requisite for CPS to save energy and increase network lifetime. As a well-known method, data aggregation can achieve energy-efficient data transmission by reducing redundant data and the amount of transmission [4]. Although there are a lot of data aggregation tree construction mechanisms for traditional networks, most of them cannot be directly applied to CPS due to the distinctive characteristics of CPS. Since the limitation on transmission power of sensor nodes creates strict constraints on the available bandwidth and the radio range of the wireless channel, the load of a node should be controlled to achieve efficient data aggregation communication [5]. In other words, the link number of the node on the data aggregation tree (i.e., node degree) should be limited to control the amount of processed dataflow of each node. Moreover, some aggregator nodes must wait to receive data from all their predecessors and then do aggregation, and this waiting causes delay. Hence, the communication delay due to transmission and aggregation can be bounded by limiting the degree of the aggregator node. The system reliability can also be enhanced by introducing the degree constraint to limit the damage caused by a single node failure. Thus, the constructed data aggregation tree should be degree-bounded [6], [7]. It should be noted that sensors may be different from each other in degree bounds, because they might have been designed using different technologies and with different goals. In consequence, the system heterogeneity (i.e., different nodes have different degree bounds) should be considered in the data aggregation tree construction. Most CPS applications perform life- and mission-critical functions, especially in target tracking, medical care, and fire monitoring, and hence must operate in real time. Consequently, real-time optimal data aggregation tree construction is mandatory to meet the real-time and power dissipation requirements of CPS applications. This paper focuses on the minimization of the maximum delay (i.e., bottleneck path weight) on a data aggregation tree with bounded degree to improve the real-time performance and the communication resource utilization, and thus all requested data can be aggregated to the data sink in the shortest time under the constraints...
of power consumption and data aggregation performance of each node. As will be proven in this paper (Theorem III.1 in Section III-A), the problem of the degree-bounded bottleneck data aggregation tree (DBBDA T) construction is NP-hard.

In this paper, we present an efficient DBBDA T construction framework based on distributed learning automata (LA) which has the ability to solve intractable NP-hard problems and can easily be implemented in a distributed manner. The network of CPS is modeled as a connected weighted and directed graph, and a learning automaton with variable action set is assigned to each node except the sink node to form a network of LA (i.e., a distributed LA). A degree-bounded data aggregation tree (DBBDA T) is constructed first by the action selection of each automaton. Then, the action vector of each automaton is updated by the linear reward-inaction \((LR - I)\) learning algorithm, and at last a DBBDA T is constructed after a certain number of iterations based on a threshold. Simulations have been conducted to evaluate the performance of the proposed framework.

This paper is organized as follows. In Section II, we summarize the related work. Section III presents the DBBDA T problem formulation and LA theory used in the proposed framework. The detailed description of the framework is given in Section IV. In Section V, we evaluate the proposed framework. Finally, we conclude this paper and outline some future work in Section VI.

II. RELATED WORK

With the growing interest in the data aggregation for CPS applications, the continual emergence of new techniques has inspired some efforts to research the efficient data aggregation infrastructure construction. Several time-efficient data aggregation techniques have already been developed [6]–[13] in real-time scenario. To combine temporal and spatial convergence of packets, Yousefi et al. [8] designed a structure-free real-time data aggregation protocol based on judiciously waiting policy and real-time data-aware anycasting policy. Du et al. [9] proposed a real time efficient data aggregation scheme with clustering routing algorithm. Abid et al. [10] presented a hybrid data aggregation method that combines multilevel clustering and structure-free algorithm for real-time WSN applications. An et al. [11] studied minimum latency data aggregation schedules in the metric and geometric signal-to-interference-noise-ratio (SINR) model. Based on the probabilistic next node selection and the neighbor information table, Mohanty et al. [12] proposed an energy efficient delay sensitive reliable transport protocol for intelligent data aggregation and forwarding. Xiao et al. [13] studied the minimum latency aggregation scheduling (MLAS) problem in WSNs under the physical interference model combined with successive interference cancellation (SIC). They formulated the problem of MLAS with SIC and proposed two heuristic polynomial-time scheduling algorithms to resolve it. However, these existing researches assume that each node can connect to all nodes within its transmission range (i.e., without considering degree constraint and platform heterogeneity). Cheng et al. [6] modeled the real-time requirement in sensor networks as the constraints on the node degree and tree height for the data aggregation tree, and presented three heuristic algorithms to build a minimum spanning tree (MST) with hop and degree constraints. Qi et al. [7] proposed a degree-based adaptive algorithm for data aggregation tree to prolong network lifetime and achieve lower latency. Both of them consider the degree constraint on each node, but they only focus on average-latency minimization, while ignoring the minimum maximum-latency problem, and thus it cannot ensure that all requested data are aggregated to the data sink in the shortest time.

Although all existing data aggregation tree construction techniques provide some solutions to improve performance and efficiency of communication in CPS, most of them do not consider both maximum-latency minimization and degree constraint. Therefore, the DBBDA T construction is still a challenging issue. LA [14] are artificial intelligence tools to find an optimal action from the action-set through repeated interactions with a random environment. It has been shown that LA have the ability to solve intractable NP-hard problems. Recently, several LA-based solutions have been proposed for solving data aggregation problem [15], [16], MST problem [17], energy-efficient topology formation problem [18], QoS problem in IaaS-based cloud systems [19], and medium access control problem in CPS [20]. Inspired by them, a DBBDA T construction framework based on LA is proposed. Major differences between this work and the aforementioned existing related researches are as follows.

1) In order to construct DBBDA T, a network of LA is formed based on the network of CPS, and each node (except the sink node) selects an optimal successor node through a learning algorithm. The proposed framework can easily be implemented in a distributed manner to maintain scalability and address some weaknesses of centralized solution.

2) Time complexity of our approach is significantly shorter than that of the integer linear programming (ILP)-based method. It can efficiently solve the problem for the sparse graph in practice. It can obtain an optimal solution or an approximation solution with guaranteed approximation ratios, and the tradeoff between accuracy and cost can be controlled by choosing the appropriate learning rate and threshold. This means that its complexity can be accommodated to the required optimality of the solution.

3) In the construction process, the degree constraint of each node is used to consider the power consumption constraints of each node and fulfill the reliability requirement of the system. Both the bottleneck path weight and the total path weight can be minimized to enhance the real-time performance of the system.

III. PRELIMINARIES

A. Degree-Bounded Bottleneck Data Aggregation Trees

In this paper, we consider a network of CPS that can be modeled as a connected weighted and directed graph \(G\) defined by a triple \(G = (V, E, W)\), where \(V = \{v_0, v_1, \ldots, v_n\}\) is a finite set of network nodes and \(v_0\) denotes the sink node, \(E =\)
Given a graph $G$ in the degree-bounded data aggregation tree, where $E$ is the cardinality of the edge set $E$. Note that, the directed edge represents the data aggregation edge rather than the communication edge, and any two nodes that are connected by one directed edge can communicate with each other.

On the top of network, a set of possible data aggregation trees $T(V) = \{\tau_1, \tau_2, \ldots\}$ rooted at sink node $v_0$ can be built. For any node $v_i \in V(i \neq 0)$, there is a unique path from $v_1$ to $v_0$ in the data aggregation tree. In other words, a data aggregation tree $\tau_i \in T(V)$ of a network is a spanning tree of the corresponding connected weighted and directed graph $G$ connecting all nodes. Given a graph $G$ and a sink node $v_0$, we are interested in finding a data aggregation tree $\tau^*$ of $G$ whose largest path weight is minimum over all data aggregation trees $T(V)$ of $G$, and the data aggregation tree $\tau^*$ is called the bottleneck data aggregation tree (BDAT). The objective criterion of BDAT problem can be formalized as

$$\min_{\tau^* \in T(V)} \max_{v_i \in \tau^*} \{w(\phi_{v_i})\} \quad \text{for all } v_i \in V - \{v_0\}$$

where $\phi_{v_i}$ is the path from $v_1$ to $v_0$ and $w(\phi_{v_i})$ is the path weight of $\phi_{v_i}$.

Obviously, BDAT problem is different from the well-known MST problem whose objective is to minimize the sum of edge weights of the spanning tree. Fig. 1 illustrates the difference between MST and BDAT rooted at $v_0$. The sums of edge weights of Fig. 1(b) and (c) are 5 and 7, respectively. On the other hand, the largest path weights of Fig. 1(b) and (c) are $\max\{3, 3+2\} = 5$ and $\max\{3, 4\} = 4$, respectively. Therefore, Fig. 1(b) is MST rather than BDAT, while Fig. 1(c) is BDAT but not MST. BDAT problem can be solved by a simple modification of Dijkstra’s algorithm [21].

However, due to energy-sensitiveness and data aggregation performance constraint, the load of a node should be controlled to achieve efficient data aggregation communication, where the load of a node corresponds to the amount of processed dataflow. Therefore, it is critical to limit the number of incoming connections (i.e., in-degree bounded in a graph terminology) in data aggregation tree. The set of degree-bounded data aggregation trees $T^{db}(V) = \{\tau_1, \tau_2, \ldots\}$ of a network is a subset of $T(V)$ which satisfies in-degree bounds of all nodes. For any network node $v_i \in V$, there is a constant $inDb(v_i) \in \{0, 1, \ldots, |E^{+}(v_i)|\}$ denoting the maximum in-degree of $v_i$ in the degree-bounded data aggregation tree, where $E^{+}(v_i)$ is the set of incoming edges to $v_i$. As shown in Fig. 2, there are three edges to $v_0$, but considering its actual capability and power constraint, it can connect only two incoming edges in the actual degree-bounded data aggregation tree. The aim of this paper is to find a BDAT satisfying in-degree bounds of all nodes, namely DBBDAT, for a given graph $G$ and a sink node $v_0$. As shown in Fig. 2, the DBBDAT problem is different from the degree-bounded minimum cost data aggregation tree (DBMDAT) problem [22], [23] which is to minimize the sum of path weight over all data aggregation trees. Note that, DBBDAT does not always exist in arbitrary strongly connected directed graphs due to degree bound. Moreover, DBBDAT problem is NP-hard as presented in Theorem III.1.

**Theorem III.1.** DBBDAT problem is NP-hard.

**Proof:** By reducing DBBDAT problem to traveling salesman problem (TSP) which is known to be NP-complete, we can prove that DBBDAT problem is NP-hard. Given a complete weighted graph $H$, TSP problem is to find a Hamilton cycle in $H$ with minimum weight (i.e., a minimum weight closed path of visiting all nodes exactly once and returning to the starting node). A graph $G$ can be created by adding a copy of sink node $v_0$ to extend graph $H$, denoted by $v_0'$, such that $e_{v_0, v_0'} = e_{v_0, v_0'}$ and $e_{v_0, v_0'} = e_{v_0, v_0'}$ for all nodes $v_i \neq v_0$. The TSP on graph $H$ can be solved by formulating DBBDAT problem on graph $G$ where $inDb(v_0) = 1$ for all nodes $v_i \neq v_0$ and $inDb(v_0') = 0$ (i.e., each node has exactly one predecessor except $v_0'$). Thus, the data aggregation tree found by DBBDAT problem is a path from node $v_0'$ to node $v_0$ with minimum weight, and this tree path corresponds to the Hamilton Cycle with minimum weight in graph $H$ (i.e., the solution of the TSP on graph $H$) while considering nodes $v_0$ and $v_0'$ as the same node. Therefore, DBBDAT problem is NP-hard.

**B. Learning Automaton**

A learning automaton is an adaptive finite state machine that interacts with an unknown random environment and tries to choose an optimal action from a finite set of allowable actions via a learning process [14]. Fig. 3 shows the learning process of learning automaton. At every instant, the automaton selects an action from the offered action set at random based on a probability vector containing the selection probability of each action. The environment evaluates the chosen action and responds with a reward or penalization feedback. Based on the response from the environment, the automaton updates the probability vector by a learning algorithm for the next action selection. Through the repetition of learning process, the automaton learns to select an action with the best response, and thus the optimal action to the environment will be chosen more frequently than other actions to minimize the average penalty and maximize the

---

[Fig. 1](#) (a) Graph $G$; (b) MST; and (C) BDAT rooted at $v_0$.  

[Fig. 2](#) (a) Graph $G$; (b) DBMDAT; and (C) DBBDAT for $inDb(v_0) = 2$. 

mean reward. In general, the environment is described by the triple \((A, R, D)\) and a learning automaton by the quadruple \((A, R, P, S)\), where

1. \(A = \{\alpha_1, \ldots, \alpha_r\}\) denotes the set of all actions of the automaton, where \(r\) is the number of actions that can be taken by the automaton. The action of the automaton at instant \(k\) is represented by \(\alpha(k)\) and \(\alpha(k) \in A\). \(A\) is the input set of the environment provided by the automaton.

2. \(R = \{\beta_1, \ldots, \beta_n\}\) represents the response set of the environment. The response at instant \(k\) is denoted by \(\beta(k) \in R\). \(R\) is the input set of the automaton received from the environment. According to the form and value scope of \(R\), the environment can be classified into P-model, Q-model, and S-model. If the response can only take two binary values 0 and 1, called a reward or non-reward, and \(\beta(k) = 1\), called a penalty, it is said to be P-model. If the response can take a finite number of values in the interval \([0, 1]\), it is called Q-model. If the response lies in the interval \([0, 1]\), it is called S-model. If the response is in the interval \([0, 1]\), the environment is referred to as S-model. In this paper, the environment is considered to be P-model.

3. \(D = \{d_1, \ldots, d_r\}\) denotes the reward probability set of the environment, where \(d_i(k) = \text{Prob}\{\beta(k) = 0|\alpha(k) = \alpha_i\}\) (i.e., the element \(d_i\) is associated with the given action \(\alpha_i\)). If reward probabilities are independent of \(k\), the random environment is said to be stationary. Otherwise, it is called a non-stationary environment. The environment used in this paper is stationary.

4. \(P = [p_1, \ldots, p_r]\) is the action probability vector. \(P(k) = [p_1(k), \ldots, p_r(k)]\) denotes the probability vector over the action set \(A\) at instant \(k\), and for all \(k\), we have \(0 \leq p_i(k) \leq 1\) and \(\sum_{i=1}^{r} p_i(k) = 1\).

5. \(S\) represents the learning algorithm which updates the action probability vector. At each instant \(k\), action probability vector \(P(k)\) is updated by linear learning algorithms given in (2) if action \(\alpha_i(k)\) is successful, and it is updated as given in (3) if the chosen action is failed.

\[
p_j(k+1) = \begin{cases} \frac{p_j(k) + a \times (1 - p_j(k))}{(1 - a) \times p_j(k)}, & j = i \\ p_j(k), & \forall j \neq i \end{cases} \quad (2)
\]

\[
p_j(k+1) = \begin{cases} \frac{1 - b}{r - 1} + (1 - b) \times p_j(k), & j = i \\ b/j \times p_j(k), & \forall j \neq i \end{cases} \quad (3)
\]

where \(a\) and \(b\) represent reward factor and penalty factor, respectively, \(a \in [0, 1]\) and \(b \in [0, 1]\). If \(a = b = 0\), \(S\) is called linear reward-penalty \((L_{R-P})\) algorithm; if \(a \gg b\), it is referred to as linear reward-\(\epsilon\) penalty \((L_{R-\epsilon-P})\) algorithm, if \(a = 0\) and \(b \neq 0\), it is called linear penalty-inaction \((L_{P-I})\) algorithm (i.e., action probability vectors are unchanged when the selected action is rewarded). According to the response from the environment, the scaled action probability vector of the automaton is updated by the DBBDAT algorithm as given in (2). If edge \((v_1, v_2)\) is selected, the available action subset \(A(k)\) is updated as follows:

\[
p_i(k+1) = p_i(k)/\sum_{\alpha_i \in A(k)} p_i(k) \quad (4)
\]

where \(\sum_{\alpha_i \in A(k)} p_i(k)\) is the sum of probabilities of actions in subset \(A(k)\).

C. Learning Automaton With Variable Action Set

Due to the degree and tree constraints on nodes, allowable actions of each node are changing from instant to instant in the DBBDAT construction process. For instance, we consider the actions of the automaton at node \(v_3\) in (a) of Fig. 2. If edge \((v_2, v_0)\) is selected, the automaton chooses an action randomly from offered actions set \(A(k)\) according to the scaled probability distribution \(P(k)\) of allowable actions on the condition that the action subset \(A(k)\) has been selected. The scaled probability of the action \(\alpha_i(k)\) belonging to the action subset \(A(k)\) at each instant \(k\) is \(P(k)\) calculated as follows:

\[
p_i(k+1) = p_i(k)/\sum_{\alpha_i \in A(k)} p_i(k) \quad (5)
\]

In the learning process of the learning automaton with variable action set, the scaled probability vector \(P(k)\) of allowable actions in \(A(k)\) is calculated as given in (4) first. Then, the automaton selects an action randomly from \(A(k)\) based on \(P(k)\). According to the response from the environment, the scaled action probability vector of the automaton is updated by the DBBDAT algorithm as given in (2). Note that, the automaton only updates the probability of actions belonging to \(A(k)\), and the probability vector of all possible actions \(\alpha_i \in A(k)\) is rescaled as follows:

\[
p_i(k+1) = p_i(k+1) \times \sum_{\alpha_i \in A(k)} p_i(k) \quad (5)
\]
A learning network can be described by a triple \( (N, A, P) \), where \( N = \{N_1, \ldots, N_n\} \) denotes the set of nodes, \( A = \{A_1, \ldots, A_n\} \) denotes the set of actions of each learning automaton in which \( A_i = \{a^1_i, \ldots, a^s_i\} \) defines the set of actions that can be taken by LA \( N_i \), and each outgoing edge \( e_{ij} \) of vertex \( v_i \in V \) in the graph corresponds to an action of the learning automaton in the action set \( A_i \). \( P = \{P_1, \ldots, P_n\} \) denotes the set of action probability vector of each learning automaton in which \( P_i = [p^1_i, \ldots, p^s_i] \) defines the probability vector over the action set \( A_i \).

Fig. 4 shows the learning process of DBDBDAT construction framework. Before diving into details of the framework, let us sketch how it works for iteration \( k \).

1) A DBDAT \( \tau_k \) is constructed for graph \( G = (V, E, W) \) and action probability vector of each learning automaton \( P_k = \{P^k_1, \ldots, P^k_n\} \) is obtained (see Section IV-B).

2) Based on the largest path weight \( \text{maxtoRoot}_k \) and the total path weight \( \omega_k \) of DBDAT \( \tau_k \), threshold of the largest path weight \( \text{maxtoRoot}_{\text{threshold}} \) and threshold of the total path weight \( \omega_{\text{threshold}} \), and the action probability vector \( P_k \) is updated (see Section IV-C).

The learning processes as described above are repeated, until choice probability \( \rho_k \) of a DBDAT exceeds a prespecified threshold \( \delta_p \) or iteration number \( k \) is greater than a certain threshold \( \delta_c \). The choice probability of a DBDAT is calculated as the product of choice probability of edges in the constructed DBDAT. For instance, the choice probability of DBDAT (b) in Fig. 2 at iteration \( k \) is equal to \( P^v_{01}(k) \cdot P^v_{v2}(k) \cdot P^v_{v3}(k) \), where \( P^v_{v_j}(k) \) is the choice probability of edge \( v_i, v_j \). The DBDAT constructed just before the algorithm stops is the DBDBDAT with minimum bottleneck path weight and total path weight among all DBDATs of the graph.

### IV. LA-BASED DBDAT CONSTRUCTION FRAMEWORK

#### A. Framework Overview

In the purpose of constructing DBDBDAT, distributed LA (i.e., a network of LA) are formed by assigning a learning automaton to each node except the sink node. The resulting network can be described by a triple \( (N, A, P) \), where \( N = \{N_1, \ldots, N_n\} \) denotes the set of nodes, \( A = \{A_1, \ldots, A_n\} \) denotes the set of actions of each learning automaton in which \( A_i = \{a^1_i, \ldots, a^s_i\} \) defines the set of actions that can be taken by LA \( N_i \), and each outgoing edge \( e_{ij} \) of vertex \( v_i \in V \) in the graph corresponds to an action of the learning automaton in the action set \( A_i \). \( P = \{P_1, \ldots, P_n\} \) denotes the set of action probability vector of each learning automaton in which \( P_i = [p^1_i, \ldots, p^s_i] \) defines the probability vector over the action set \( A_i \).

#### B. LA-Based DBDAT Formation

To construct DBDBDAT with learning processes, DBDAT should be formed first. We first give the definitions of some variables used in the algorithm, as listed in Table I. Algorithm 1 shows the DBDAT formation algorithm. The data aggregation tree is rooted at the sink node \( v_0 \) and grows until the tree spans all vertices in \( V \). Here, \( Q \) denotes the set of nodes that have connected to \( v_0 \), and \( U \) denotes the set of nodes that can connect to \( v_0 \) via nodes in set \( Q \).

1. A DBDAT \( \tau_k \) is constructed for graph \( G = (V, E, W) \) and action probability vector of each learning automaton \( P_k = \{P^k_1, \ldots, P^k_n\} \) is obtained (see Section IV-B).

2. Based on the largest path weight \( \text{maxtoRoot}_k \) and the total path weight \( \omega_k \) of DBDAT \( \tau_k \), threshold of the largest path weight \( \text{maxtoRoot}_{\text{threshold}} \) and threshold of the total path weight \( \omega_{\text{threshold}} \), and the action probability vector \( P_k \) is updated (see Section IV-C).

The learning processes as described above are repeated, until choice probability \( \rho_k \) of a DBDAT exceeds a prespecified threshold \( \delta_p \) or iteration number \( k \) is greater than a certain threshold \( \delta_c \). The choice probability of a DBDAT is calculated as the product of choice probability of edges in the constructed DBDAT. For instance, the choice probability of DBDAT (b) in Fig. 2 at iteration \( k \) is equal to \( P^v_{01}(k) \cdot P^v_{v2}(k) \cdot P^v_{v3}(k) \), where \( P^v_{v_j}(k) \) is the choice probability of edge \( v_i, v_j \). The DBDAT constructed just before the algorithm stops is the DBDBDAT with minimum bottleneck path weight and total path weight among all DBDATs of the graph.

### TABLE I. VARIABLES AND NOTATIONS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_0 )</td>
<td>Sink node</td>
</tr>
<tr>
<td>( Q )</td>
<td>Set of nodes that have connected to the sink node</td>
</tr>
<tr>
<td>( U )</td>
<td>Set of nodes that can connect to the sink node via nodes in the set ( Q )</td>
</tr>
<tr>
<td>( v^a )</td>
<td>The node selected randomly from the set ( U )</td>
</tr>
<tr>
<td>( v^p )</td>
<td>A direct successor of ( v_q ) that is selected from the set ( Q ) based on the probability vector</td>
</tr>
<tr>
<td>( v^{'\text{toRoot}} )</td>
<td>The weight from the node ( v^\prime ) to the sink node</td>
</tr>
<tr>
<td>( \omega_k )</td>
<td>Maximum path weight from nodes in set ( U ) to the sink node at iteration ( k )</td>
</tr>
<tr>
<td>( \omega_{\text{available}} )</td>
<td>The incoming connectable number of the node ( v^\prime )</td>
</tr>
</tbody>
</table>

Fig. 2 at iteration \( k \) is equal to \( P^v_{01}(k) \cdot P^v_{v2}(k) \cdot P^v_{v3}(k) \), where \( P^v_{v_j}(k) \) is the choice probability of edge \( v_i, v_j \). The DBDAT constructed just before the algorithm stops is the DBDBDAT with minimum bottleneck path weight and total path weight among all DBDATs of the graph.
Algorithm 1. DBDAT Formation Algorithm

Require: Graph \((G), \text{Sink node } (v_0), \text{Probability vector } (p)_k^{k-1}\). Initially \(\text{maxtoRoot}_k = 0, \omega_k = 0, Q = \Phi, U = \Phi\).

Ensure: Selected edges of DBDAT \((G_k)\), Maximum path weight \((\text{maxtoRoot}_k)\), Total path weight \((\omega_k)\).

1: \(Q = Q.\text{Add}(v_0)\);
2: \(U = U.\text{Union}(v_0, \text{ predecessors})\);
3: for each node \(\beta \in v_0.\text{ predecessors} \) do
4: \(\beta.\text{successors}.\text{Add}(v_0)\);
5: end for
6: while \(U \neq Q\) do
7: \(v^s = \text{Random}(U)\);
8: \(U = U.\text{Remove}(v^s)\);
9: \(Q = Q.\text{Add}(v^s)\);
10: if \(v^s.\text{ available} > 0\) then
11: for each node \(\lambda \in (v^s.\text{ predecessors} - Q)\) do
12: \(\lambda.\text{successors}.\text{Add}(v^s)\);
13: end for
14: \(U = U.\text{Union}(v^s.\text{ predecessors} - Q)\);
15: end if
16: \(v^p = \text{RandomWithProbability}(v^s.\text{successors}, p)_k^{k-1}\);
17: \(G_k.\text{edge}[v^s][v^p].\text{selected} = \text{TRUE}\);
18: \(v^p.\text{ available} = -1\);
19: if \(v^p.\text{ available} == 0\) then
20: for each node \(\gamma \in (v^p.\text{ predecessors} - Q)\) do
21: \(\gamma.\text{successors} = \gamma.\text{successors}.\text{Remove}(v^p)\);
22: if \(\gamma.\text{successors} == \Phi\) then
23: \(U = U.\text{Remove}(\gamma)\);
24: end if
25: end for
26: end if
27: \(v^s.\text{toRoot} = v^p.\text{toRoot} + G_k.\text{edge}[v^s][v^p].\text{weight}\);
28: \(\omega_k = \omega_k + v^s.\text{toRoot}\);
29: if \(v^s.\text{toRoot} > \text{maxtoRoot}_k\) then
30: \(\text{maxtoRoot}_k = v^s.\text{toRoot}\);
31: end if
32: end while

as given in Table II. In Fig. 5(a), node \(v_0\) is considered as the sink node and it is added into set \(Q\).

In this case, nodes \(v_1, v_2, v_3, v_5\) are added into set \(U\), because they can connect to node \(v_0\) directly. In Fig. 5(b), node \(v_3\) is selected as \(v^s\), and \(v_0\) is selected as its successor node (i.e., \(v^p\)). Thus, node \(v_3\) is added into set \(Q\) and removed from set \(U\). In Fig. 5(c), node \(v_1\) is selected as \(v^s\) and \(v_0\) is selected as its successor node. At the same time, node \(v_4\) is added into set \(U\), since it can connect to \(v_0\) through \(v_1\). In Fig. 5(d), node \(v_4\) is selected as \(v^s\) and \(v_2\) is selected as its successor node. In Fig. 5(e), node \(v_2\) is selected as \(v^s\) and \(v_3\) is selected as its successor node. In Fig. 5(f), node \(v_5\) is selected as \(v^s\) and \(v_2\) is selected as its successor node. Up to this step, we have set \(Q\) of \(v_0, v_1, v_2, v_3, v_4, v_5\) and set \(U\) of \(\Phi\). The entire network is connected because the size of set \(Q\) is equal to the number of nodes in the graph. Thus, we can obtain the results described in Table II. As can be found from the fifth column (i.e., \(v^p\)) of Table II, the incoming connection of each node is not more than

<table>
<thead>
<tr>
<th>NO.</th>
<th>(Q)</th>
<th>(U)</th>
<th>(v^s)</th>
<th>(v^p)</th>
<th>Selected edges</th>
<th>(\text{maxtoRoot}_k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(v_0)</td>
<td>(v_1, v_2, v_3, v_5)</td>
<td>(v_0)</td>
<td>(v_0)</td>
<td>(v_3, v_0)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>(v_0, v_3)</td>
<td>(v_1, v_2, v_3, v_5)</td>
<td>(v_3)</td>
<td>(v_1)</td>
<td>(v_3, v_0)</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>(v_0, v_1, v_3)</td>
<td>(v_1, v_2, v_4, v_5)</td>
<td>(v_1)</td>
<td>(v_1)</td>
<td>(v_3, v_0)</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>(v_0, v_1, v_3)</td>
<td>(v_2, v_4, v_5)</td>
<td>(v_4)</td>
<td>(v_1)</td>
<td>(v_4, v_1)</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>(v_0, v_1, v_3)</td>
<td>(v_5)</td>
<td>(v_2)</td>
<td>(v_1)</td>
<td>(v_4, v_1)</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>(v_0, v_1, v_3)</td>
<td>(v_2, v_3, v_4, v_5)</td>
<td>(v_5)</td>
<td>(v_5)</td>
<td>(v_2, v_3)</td>
<td>7</td>
</tr>
</tbody>
</table>

2. In other words, the degree constraint of each node is satisfied in the process of DBDAT formation.

C. Learning Algorithm

To show how the learning algorithm updates the probability vector, let us assume that DBDAT \(\tau_k\) is selected at stage \(k\). At each stage \(k > 1\), the minimum bottleneck path weight of all the constructed DBDATs until stage \(k\) is calculated as the dynamic threshold by utilizing (6).

\[
\xi_k = \min\{\xi_{k-1}, \text{maxtoRoot}_k\}
\]  

where \(\xi_{k-1}\) is the minimum bottleneck path weight of all the constructed DBDATs until stage \(k - 1\) and \(\text{maxtoRoot}_k\) is the bottleneck path weight of \(\tau_k\). To construct DDBDAT based on LA theory, the minimum bottleneck path weight is needed, and thus in the learning process, at each stage, the bottleneck path weight of the selected DBDAT is compared with the dynamic threshold. If the current bottleneck path weight is less than its threshold, both bottleneck path weight threshold and overall path weight threshold are reset based on the current constructed data aggregation tree, and the reward algorithm as shown in Algorithm 2 is called to update the probability vector. Algorithm 2 is realized based on (2) and (5), where \(\tau_k\) is the DBDAT constructed at iteration \(k\),
Algorithm 2. Reward Algorithm.

Ensure: The updated probability vector \((P^{k+1})_\beta\).

1: for each node \(\beta\) in \(\tau_k\).nodes do
2: \(p^\text{sum}_\beta = 0\);\n3: for each node \(\lambda\) in \(\tau_k\).successors(\(\beta\)) do
4: \(p^\text{sum}_\beta = p^\text{sum}_\beta + \tau_k\).edge[\(\beta\)]\([\lambda]\).probability;\n5: end for
6: end for
7: end for

\(P^k\) is the probability vector at iteration \(k\), and \(\sigma\) is the learning rate used in the learning algorithm. Moreover, to minimize the total path weight (i.e., the average path weight), the current bottleneck path weight is equal to its threshold, while the current overall path weight is less than its threshold, the overall path weight threshold is reset, and the probability vector is updated by calling the reward algorithm. It is obvious that both bottleneck path weight and overall path weight are minimized by the learning algorithm in the DBBDAT construction process.

In the distributed implementation of the algorithm, the sink node \(v_0\) just needs to control the activation and termination of the learning process and the random selection of \(v^\alpha\). For any selected node \(v^\alpha\), the selection of \(v^\alpha\) and the update of its action probability vector (i.e., the reward algorithm) can be done by itself through the message switching with its successors. Thus, it can easily be implemented in a distributed manner. Due to space constraints, we omit the detailed distributed implementation of the variable action-set LA here and refer the reader to [25] for details.

D. Algorithm Analysis

In this section, we will theoretically prove the optimality of our algorithm, and analyze its computation complexity.

Definition IV.1. [26] A learning automaton is said to be absolutely expedient if
\[
E\left[M(k + 1)\big| P(k)\right] < M(k)
\]
for each stage \(k\), and each penalty probability \(c_i\) for each action \(\alpha_i \in A\). \(M(k)\) is the expected penalty at stage \(k\) from the environment, and it is defined as
\[
M(k) = E\left[\beta(k)\big| P(k)\right] = \sum_{i=1}^{r} p_i(k) c_i
\]
where the penalty probability \(c_i\) associated with action \(\alpha_i \in A\) is defined as
\[
c_i = \text{Prob}\{\beta(k) = 1|\alpha(k) = \alpha_i\} = 1 - d_i
\]
with probability 1 as \(k \to \infty\) can be obtained for any arbitrary \(\epsilon > 0\) by a suitable choice of learning parameters.

Formally, a learning automaton is absolutely expedient if the expected payoff, computed with the revised probability of choosing each strategy, is greater than the expected payoff with previous probabilities. \(\epsilon\)-optimality of a learning automaton implies that the performance of the automaton can be made as close to optimal as desired. We can prove that the automaton with variable action set for DBBDAT construction is absolutely expedient (Lemma IV.1) and also \(\epsilon\)-optimal (Lemma IV.2) by using \(LR - I\) learning algorithm. The detailed proof for Lemmas IV.1 and IV.2 can be found in Appendices A and B.

Lemma IV.1. The automaton with variable action set for DBBDAT construction is absolutely expedient by using the linear reward-inaction \((LR - I)\) learning algorithm.

Lemma IV.2. The automaton with variable action set for DBBDAT construction is \(\epsilon\)-optimal by using the linear reward-inaction \((LR - I)\) learning algorithm.

Theorem IV.1. The total time complexity of the LA-based DBBDAT construction algorithm is \(O(Tm)\), where \(T\) is the iteration number and \(m\) is the edge number.

Proof: Let \(n\) and \(m\) denote the number of nodes and edges in the network of CPS, respectively, so the average out-degree of each node is \(O(m/n)\). From the algorithm overview given in Section IV-A, we can find that the main time complexity of DBBDAT construction algorithm lies in DBBDAT construction and action probability vector updating. Since the number of actions for each learning automaton is not more than the out-degree of the node, DBBDAT construction needs \(O(m/n) \times O(n) = O(m)\) time to choose actions for all LA. For each node, it takes \(O(m/n)\) time to update its action probability vector, so the action probability vector updating needs \(O(m/n) \times O(n) = O(m)\) time for all LA. Therefore, the total time complexity of the LA-based DBBDAT construction algorithm is \(O(Tm)\), where \(T\) is the iteration number.

Remarks. A graph is said to be sparse if its total number of edges is small compared to the total number of possible edges. Formally, in a sparse graph, \(m = O(n)\). We note that for sparse graphs, the total time complexity of our algorithm can be reduced to \(O(Tn)\) based on Theorem IV.1. That is, our algorithm is relatively more effective for sparse graphs, and it is validated by the experiment. The experimental results show that it takes less than 5 min for the network with less than 150 nodes randomly deployed into a region of 200 m \times 200 m and the transmission radius of each node is fixed to 25 m. Therefore, our algorithm can efficiently solve the DBBDAT problem for the sparse graph in practice.
TABLE III.
NETWORK TOPOLOGY INSTANCES

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>7</td>
<td>32 (75%)</td>
</tr>
<tr>
<td>G2</td>
<td>7</td>
<td>42 (100%)</td>
</tr>
<tr>
<td>G3</td>
<td>8</td>
<td>42 (75%)</td>
</tr>
<tr>
<td>G4</td>
<td>8</td>
<td>56 (100%)</td>
</tr>
<tr>
<td>G5</td>
<td>10</td>
<td>68 (75%)</td>
</tr>
<tr>
<td>G6</td>
<td>10</td>
<td>90 (100%)</td>
</tr>
</tbody>
</table>

V. EXPERIMENTAL RESULTS

The performance of the proposed DBBDAT construction framework based on LA is evaluated by using NetworkX [27] which is a Python language software package for the study of the structure, dynamics, and functions of complex networks. The simulations run on a computer with Intel (R) Core (TM) i5 2.8-GHz CPU and 4-GB memory.

In order to compare the performance of our approach with the ILP-based method, as shown in Table III, six sets of random instances of network topology graphs are generated to study the algorithm scalability. For the graph with vertex number from set \{7, 8, 10\}, the edge creation probabilities for two different vertices are 75% and 100%. The in-degree bound of each vertex is chosen from a discrete uniform distribution on interval \((0, n/2)\), where \(n\) is vertex number. A random integer is selected from \((1, 10)\) which is associated with each edge as the edge weight. To evaluate the practicability and scalability of our approach on large-scale random networks, we randomly deploy nodes into a region of \(200 \text{ m} \times 200 \text{ m}\), and the transmission radius of each node is fixed to 25 m. For each node, its in-degree bound is chosen from a discrete uniform distribution on interval \((0, \Delta)\), where \(\Delta\) is its in-degree in random graphs.

The convergence of the LA-based algorithm is evaluated first to demonstrate its correctness and effectiveness. Then, the impact of the learning rate and the threshold on the number of required iterations and the result accuracy is studied in the experiment to make a tradeoff between cost and solution optimality by choosing appropriate learning rate and threshold. Moreover, we compare our approach with the ILP-based method in terms of time complexity, and for the detailed ILP formulation, interested readers are referred to [28]. For the ILP-based algorithm, GNU linear programming kit (GLPK) solver [29] and PuLP modeler [30] in Python are used to solve the ILP formulation. At last, we evaluate the average solution time and average accuracy of our approach on large-scale random networks to verify its practicability and scalability. We repeated each simulation multiple times, and observed similar results in each trial. In the following, we present a representative result from one trial and analyze it in detail.

Fig. 6 shows the evolution of probability associated with optimal actions for the graph of Fig. 5(a), where the learning rate is set to 0.15. We can find that at the beginning of the experiment, the probability has some fluctuation, and then gradually increases. Not surprisingly, at last, the probability converges to 1 after about 200 iterations. Note that, some actions with minor probability are chosen in the process of the experiment due to the random action selection of each automaton based on its probability vector, but this does not affect the correctness of the framework, because the algorithm is based on the threshold.

Fig. 8 shows the iteration number for different learning rates versus thresholds for graphs \(G_1, G_2, G_3, G_4, G_5,\) and \(G_6\). It can be found that the iteration number increases as the learning rate reduces and the threshold increases, and vice versa. Moreover, it also increases with the increase in the number of nodes and edges. The corresponding accuracy of the LA-based algorithm for different learning rates and thresholds is illustrated in Fig. 9. We can find that the accuracy is volatile for larger learning rates and smaller thresholds, and this fluctuation reduces with the increase in the number of nodes and edges for the same learning rate and threshold. Therefore, based on the results given in Figs. 8 and 9, it can be concluded that the number of required iterations and the solution accuracy are inversely proportional to the learning rate, while directly proportional to the threshold. Thus, the complexity of the LA-based approach can be accommodated to the required optimality of the solution by choosing appropriate learning rate and threshold. We can also find that the solution accuracy is greater than the probability of choosing the constructed DBDAT (i.e., the threshold) for all learning rates and network topologies, and thus the approximation ratio of the solution can be bounded by the threshold. Therefore, an approximation solution with a guaranteed approximation ratio can be obtained based on a specific threshold. In addition, for a specific threshold, the more edges that the network topology has, the better the accuracy of the solution.
It can be found from Fig. 8(a) that the number of required iterations for learning rates smaller than 0.1 is significantly larger than those for learning rate larger than 0.1 for $G_1$ when the threshold is 0.8. In addition, from Fig. 9(a), we can find that the result accuracy for learning rates larger than 0.1 is the same as that for the learning rate of 0.1 for $G_1$ when the threshold is 0.8. Hence, we can conclude that a tradeoff between the result accuracy and the cost of the algorithm (the iteration number) can be made when the learning rate is set to 0.1 and the threshold is 0.8. So, the learning rate and the threshold are set to 0.1 and 0.8, respectively, for $G_1$ in the further experiment. Similarly, the learning rate and the threshold of other graphs are determined based on results given in Figs. 8 and 9, as shown in Table IV.

![Fig. 8. Iterations of different graphs for different learning rates and thresholds. (a) $G_1$ and $G_2$. (b) $G_3$ and $G_4$. (c) $G_5$ and $G_6$.](image)

![Fig. 9. Accuracy of different graphs for different learning rates and thresholds. (a) $G_1$ and $G_2$. (b) $G_3$ and $G_4$. (c) $G_5$ and $G_6$.](image)

Fig. 10. Time consumption for different graphs.

**TABLE IV**

<table>
<thead>
<tr>
<th>Graph</th>
<th>Learning rate</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>0.1</td>
<td>0.8</td>
</tr>
<tr>
<td>$G_2$</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>$G_3$</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>$G_4$</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>$G_5$</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>$G_6$</td>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Figs. 11 and 12 show the average solution time and the average accuracy for different numbers of nodes when the learning rate is fixed to 0.4, respectively. Results show that the average time consumption of LA- and ILP-based algorithms for different graphs to obtain a DBBDAT increases as the number of nodes and edges increases. This is because the search space increases due to the increase in nodes and edges, and so more iterations or integer variables are required to explore possible solutions. We can also find that the time consumed by the LA-based algorithm is dramatically shorter than that of the ILP-based algorithm, especially for $G_3$, $G_4$, $G_5$, and $G_6$. The reason for this is that the DBBDAT problem of graph $G_3$ with 8 nodes and 42 edges has 336 integer variables, which exceeds the upper limit of the number of integer variables of the ILP solver of GLPK, and thus the efficiency of the ILP-based algorithm reduces rapidly, while such effect on the LA-based algorithm is relatively small.
solution time increases with the node number and the threshold; however, the formulated problem can be solved efficiently (in less than 5 min) with LA-based algorithm when there are no more than 150 nodes, and thus network clustering can be used to alleviate the scalability problem. We can also find that the average accuracy is above 90% for all thresholds. It is interesting to see that the average accuracy decreases as the node number increases when the node number is not greater than 120, and then it grows with the increase in the node number. The reason for this is that, when the node number is not greater than 120, the random graph is sparse, and thus the search space increases due to the increase in nodes and edges. While there are more than 120 nodes, the random graph is relatively dense, and thus the number of DBBDATs for a graph increases as the node number increases (i.e., there may be multiple DBBDATs with the same minimum maximum-latency for a graph). Hence, a DBBDAT may be constructed with a lower threshold for a dense graph.

From the above-described simulation results, we can conclude that in the LA-based framework, the probability associated with optimal actions converges to 1 after a few iterations where each automaton selects a suitable action to solve the DBBDAT problem. Compared with the ILP-based method, our approach not only can improve the time consumption but also can make a tradeoff between accuracy and computation overhead to get an optimal solution or an approximation solution with guaranteed approximation ratios by choosing appropriate learning rate and threshold. In addition, the efficiency of the ILP-based method is seriously subject to the restriction of integer variables of the ILP solver of GLPK, while the LA-based approach is affected little by this limit, and thus it is more scalable. Moreover, our approach can efficiently construct DBBDAT for the sparse graph in practice. At last, our approach can be easily implemented in a distributed manner to maintain scalability and address some weaknesses of centralized solution.

VI. CONCLUSION

In this paper, we formulated the DBBDAT problem in CPS and presented an LA-based solution, which has not been reported in any previous work. The LA-based framework can discover the bottleneck data aggregation tree topology with in-degree constraint to achieve energy-efficient real-time data transmission by reducing the amount of transmission data and minimizing the maximum delay. We demonstrate the correctness and the effectiveness of the algorithm by extensive experiments. Simulation results show that our approach achieves high performance. The primary contributions of this paper are summarized as follows.

1) An LA theoretic framework is adopted, in which each node just interacts with its predecessors and successors, and thus it can be easily implemented in a distributed manner to maintain scalability and address some weaknesses of centralized solutions.

2) An optimal solution or an approximation solution with guaranteed approximation ratios can be obtained, and the tradeoff between accuracy and overhead can be controlled by choosing an appropriate learning rate and a suitable threshold.

3) In the construction process, the degree constraint of each node is used to control the amount of processed dataflow of each node and to fulfill the system reliability requirement. Moreover, the maximum delay and the average delay can be minimized in the constructed data aggregation tree to enhance the real-time performance of the system.

In terms of future work, we will consider the DBBDAT construction in the network topology with a connectivity probability for each link to enhance the system reliability. We also would like to implement our framework on real-life large-scale CPS in a distributed manner to further benchmark our framework based on more meta-heuristic algorithms and more metrics, and revise it. Moreover, it has been shown in many cases that the use of multiagent reinforcement learning can alleviate the scalability problem. This serves as another interesting direction for further investigation.

ACKNOWLEDGMENT

The authors would like to thank Prof. J. Deng at the University of North Carolina at Greensboro, America, for the insights they have gained in working with him.
**APPENDIX A: PROOF OF LEMMA IV.1**

**Proof:** For any iteration \( k \), let \( A(k) = A_j \) be the action subset at iteration \( k \). Based on (4), if \( \alpha_i \in A(k) \), we have

\[
E[p_i'(k + 1) - p_i'(k) - p_i(k)] = \sigma(1 - p_i'(k))p_i'(k)d_i - \sigma p_i(k)d_g = \sigma p_i'(k) \sum_{g \neq i, \alpha_g \in A(k)} p_g'(k)(d_i - d_g)
\]

where \( d_i \) is the reward probability for \( \alpha_i \). Based on (4), since \( A(k) = A_j \), for all \( q \) and \( \alpha_q \in A(k) \), we can get

\[
p_q'(k) = p_q(k)/K_j(k)
\]

where \( K_j(k) \triangleq \sum_{q, \alpha_q \in A_j} p_q(k) \). Thus, (11) can be rewritten as

\[
E[p_i'(k + 1) - p_i'(k) - p_i(k)] = \frac{\sigma}{[K_j(k)]^2} p_i(k) \sum_{g \neq i, \alpha_g \in A(k) = A_j} p_g(k)(d_i - d_g).
\]

Let \( r \) be the number of available actions of the automaton, the number of action subset is \( W = 2^r - 1 \). Hence, we can get

\[
E[p_i(k + 1) - p_i(k)] = \sum_{j=1}^{W} E[p_i(k + 1) - p_i(k)] = \sum_{j=1}^{W} \text{Prob}[A(k) = A_j]
\]

\[
E[p_i(k + 1) - p_i(k)] = K_j(k) \times E[p_i'(k + 1) - p_i'(k)]
\]

Based on (14) and (15), we can get

\[
E[p_i(k + 1) - p_i(k)] = \sum_{j=1}^{W} q_j(k)
\]

\[
\times K_j(k) \times E[p_i'(k + 1) - p_i'(k)]
\]

For all single-action subsets and others that do not contain action \( \alpha_i \), \( E[p_i'(k + 1) - p_i'(k)] = 0 \) Thus, (16) can be rewritten as

\[
E[p_i(k + 1) - p_i(k)] = \sum_{j=r+1}^{W} q_j(k)
\]

\[
\times K_j(k) \times E[p_i'(k + 1) - p_i'(k)]
\]

where the summation excludes the first \( r \) single-action subsets without action \( \alpha_i \). Based on (13) and (17), we have

\[
E[p_i(k + 1) - p_i(k)] = \sigma \sum_{j=r+1}^{W} q_j(k) K_j(k) p_i(k) \sum_{g \neq i, \alpha_g \in A_j} p_g(k)(d_i - d_g).
\]

Let \( M(k) = \sum_{i=1}^{r-1} (1 - d_i)p_i(k) \), and \( \Delta M(k) = E[M(k + 1) - M(k)|p(k)] \), we can get

\[
\Delta M(k) = \sum_{i=1}^{r-1} (1 - d_i)E[p_i(k + 1) - p_i(k)]p(k)
\]

\[
= \sigma \sum_{i=1}^{r} \sum_{j=r+1}^{W} q_j(k) K_j(k) p_i(k) \sum_{g \neq i, \alpha_g \in A_j} p_g(k)(1 - d_i)(d_i - d_g).
\]

Thus, \( \Delta M(k) \) can be expressed in quadratic form [31], i.e.,

\[
\Delta M(k) = p(k)C(k)p^T(k)
\]

where \( p(k) = [p_1(k), p_2(k), \ldots, p_r(k)] \), and \( C(k) \) is a \( r \times r \) symmetric matrix with

\[
c_{i,i} = 0
\]

\[
c_{i,j} = -\sigma(d_i - d_j)^2 \sum_{m=r+1}^{W} q_m(k)
\]

where \( m \) is such that \( \alpha_i \in A_m \) and \( \alpha_j \in A_m \). Since at least one of action subsets containing more than one action has a nonzero probability of being selected, \( q_m(k) \neq 0 \) for at least one subset \( A_m \), where \( r + 1 \leq m \leq 2^r - 1 \). Thus, for all nontrivial environments \( (d_i \neq d_j, \text{for } i \neq j) \), \( C(k) \) has some negative elements. Thus, if the initial action probabilities lie in the open simplex, the quadratic form \( \Delta M(k) = p(k)C(k)p^T(k) < 0 \), and hence the automaton is absolutely expedient according to the definition of absolute expediency for the learning automaton.

**APPENDIX B: PROOF OF LEMMA IV.2**

**Proof:** According to the LR – I learning algorithm used in Algorithms 1 and 2 for DBBDA construction, the complete set of actions of each automaton is selected with a positive probability at all iterations. Thus, there exists a collection of action subsets, each of which contains an optimal action, and whose union is the set of all actions (i.e., \( A \)). Let \( A_h \) denote a subset of \( A \), for each subset \( A_h \), at each iteration \( k \), the probability \( q_h(k) \) of selecting the action subset \( A_h \) meets the condition \( q_h(k) \geq \epsilon_k > 0 \). That is, each subset \( A_h \) is selected with a positive probability at all iterations, and thus it can ensure that each subset in the collection is selected infinitely. In each subset of the collection, the scaled action probability \( p_L'(k) \) of the optimal action \( \alpha_L \) tends to unity with probability \( 1 - \epsilon \). Because the collection includes all actions in \( A \), \( p_L'(k) \) emerges with a unity value when it is updated along with every other action probability of actions in \( A \). Hence, all scaled action probabilities \( p_i'(k) \) (\( i \neq L \)) tend to zero, and thus the unscaled action probability \( p_L(k) \rightarrow 1 \) with a probability arbitrarily close to unity. Therefore, the automaton with variable action set for DBBDA construction is \( \epsilon \)-optimal by using the LR – I learning algorithm.

**REFERENCES**


Feng Xia (M’07–SM’12) received the B.Sc. and Ph.D. degrees in control science and engineering from Zhejiang University, Hangzhou, China, 2001 and 2006, respectively.

He was a Research Fellow with Queensland University of Technology, Brisbane, QLD, Australia. He is currently a Full Professor with the School of Software Technology, Dalian University of Technology, Dalian, China. He has authored/coauthored 1 book and over 180 scientific papers in international journals and conferences. His research interests include social computing, mobile computing, and cyber physical systems.

Dr. Xia is a Senior Member of ACM and ACM SIGWEB. He is the (Guest) Editor of several international journals. He serves as the General Chair, PC Chair, Workshop Chair, Publicity Chair, or PC Member of a number of conferences.

Mohammad S. Obaidat (S’85–M’86–SM’91–F’05) received the M.S. and Ph.D. degrees in computer engineering, minor in computer science, from Ohio State University, Columbus, OH, USA, 1983 and 1986, respectively.

He is currently a Full Professor of Computer Science with Monmouth University, West Long Branch, NJ, USA. He has authored over 15 books and over 560 refereed technical papers.

Dr. Obaidat is a Fellow of the Society for Modeling and Simulation International. He is the Editor-in-Chief of three scholarly journals and is also an Editor or Advisory Editor of numerous international journals. He was a Chair of the numerous international conferences and given numerous keynote speeches all over the world.