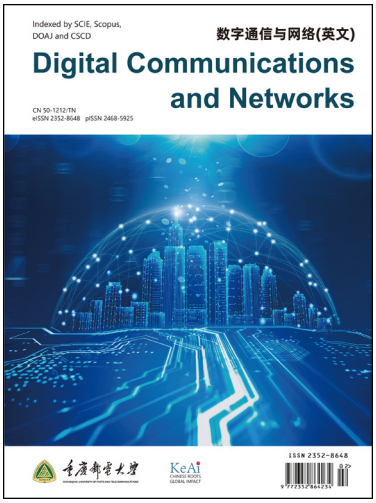
Journal Pre-proof



A learning automata based edge resource allocation approach for IoT-enabled smart cities

Sampa Sahoo, Kshira Sagar Sahoo, Bibhudatta Sahoo and Amir H. Gandomi

PII: S2352-8648(23)00173-6
DOI: https://doi.org/10.1016/j.dcan.2023.11.009
Reference: DCAN 706

To appear in: *Digital Communications and Networks*

Received date: 8 October 2021
Revised date: 12 November 2023
Accepted date: 27 November 2023

Please cite this article as: S. Sahoo, K.S. Sahoo, B. Sahoo et al., A learning automata based edge resource allocation approach for IoT-enabled smart cities, *Digital Communications and Networks*, doi: https://doi.org/10.1016/j.dcan.2023.11.009.

Digital Communications and Networks(DCN)

HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

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

A learning automata based edge resource allocation approach for IoT-enabled smart cities

Sampa Sahoo*[a], Kshira Sagar Sahoo[b], Bibhudatta Sahoo[c], Amir H. Gandomi*[d]

[a]**Department of Computer Science and Engineering, C.V. Raman Global University, Bhubaneswar, India**
[b]**Department of Computer Science and Engineering, SRM University, Amaravati, India**
[b] **Department of Computing Science, Umeå University, Umeå, 90187, Sweden**
[c]**Department of Computer Science and Engineering, NIT Rourkela, India**
[d]**Faculty of Engineering and Information Technology, University of Technology Sydney, Australia**
[d] **University Research and Innovation Center (EKIK), Óbuda University, 1034 Budapest, Hungary**

Abstract

The development of the Internet of Things (IoT) technology leading to a new era of smart applications such as smart transportation, buildings, and smart homes. Moreover, these applications act as the building blocks of IoT-enabled smart cities. The high volume and high velocity of data generated by various smart city applications are sent to flexible and efficient cloud computing resources for processing. However, there is a high computation latency due to the presence of a remote cloud server. Edge computing, which brings the computation close to the data source is introduced to overcome this problem. In an IoT-enabled smart city environment, one of the main concerns is to consume the least amount of energy while executing tasks that satisfy the delay constraint. An efficient resource allocation at the edge is helpful to address this issue. In this paper, an energy and delay minimization problem in a smart city environment is formulated as a bi-objective edge resource allocation problem. First, we presented a three-layer network architecture for IoT-enabled smart cities. Then, we designed a learning automata-based edge resource allocation approach considering the three-layer network architecture to solve the said bi-objective minimization problem. Learning Automata (LA) is a reinforcement-based adaptive decision-maker that helps to find the best task and edge resource mapping. An extensive set of simulations is performed to demonstrate the applicability and effectiveness of the LA-based approach in the IoT-enabled smart city environment.

KEYWORDS:
Edge computing, IoT, Learning automata, Resource allocation, Smart city

Note: This work is an extension of the conference paper [1], where we solved the edge resource allocation problem in an IoT-enabled smart city environment with an auction-based approach. In that work, we jointly minimized the energy consumption and the computation time of the delay-sensitive tasks. However, in the current research, we used a Learning Automata (LA)-based approach for resource allocation as a bi-objective minimization problem. The LA makes the best decision (selection of the best action) from a set of possible actions. Here, LA is considered for the edge resource allocation problem. Accordingly, the best VM and task pair are chosen to satisfy certain system constraints in a dynamic edge environment. A series of experiments have been conducted in terms of energy consumption, average delay, and success ratio to confer the effectiveness of the proposed solution. An experiment is also conducted to choose the appropriate reward and penalty values along with an ANOVA test to determine the influence of the proposed approach.

1. Introduction

Nowadays Internet of Things (IoT) becomes an indispensable part of people's life. This is because of the

*S. Sahoo, A. H. Gandomi (Corresponding author) (email:sampaa2004@gmail.com, gandomi@uts.edu.au).
[1]K. S. Sahoo, (email:ksahoo@cs.umu.se)
[2]B.D. Sahoo, (email: bibhudatta.sahoo@gmail.com).

presence of IoT technology in every aspect of our life starting from smart homes to smart healthcare, emergency services, and many more. All these smart applications are leading to a new era of IoT-enabled smart cities. In addition, the IoT-enabled applications generate a large volume of data that requires a good processing infrastructure. The other requirements of IoT-enabled smart city applications are low latency, energy efficiency, etc. For instance, an application with frequent interaction with the data source and processing unit requires low latency (e.g., smart transportation). Cloud computing is a computing paradigm that provides users with high computation and storage capacity. Cloud deployment is beneficial for IoT-enabled smart city applications [2]. Cloud-based deployment helps aggregate the data from heterogeneous IoT devices, that are present in geographically distributed areas. Although the cloud is essential for IoT deployments, it is cost-inefficient and suffers from high end-to-end latency and communication overhead [3]. This is due to the presence of geographically remote heterogeneous cloud servers that are far from the data source [4, 5]. The problems associated with cloud deployment for IoT applications affect the delay-sensitive tasks and IoT sensing devices with limited resources. In this context, edge computing technology acts as a prominent solution by bringing the computation and storage closer to the data source,i.e., IoT devices [6].

Edge computing is a layer between the IoT/device layer (data source) and the cloud layer (computing infrastructure), that keeps the data processing closer to the IoT devices. However, the edge layer has fewer resources compared to the cloud layer. These limited edge resources need to be utilized in a better and more energy-efficient way. For example, inefficient resource utilization in an autonomous car application causes a delay in data analysis and decision making, resulting in a car crash. The solution to overcome these problems is an efficient edge resource allocation method. In this context, this paper presents a Learning Automata (LA) based solution for edge resource allocation in IoT-enabled smart cities. LA uses the reinforcement-based learning method to learn from the action taken in a dynamic environment [7, 8, 9]. The continuous interaction with the working environment helps LA to make the best decision (choose the best action) from a set of possible actions. Similarly, in the edge resource allocation problem, we need to find the best mapping between a task and a computing resource (here, Virtual Machine (VM) in edge layer) from a set of possible pairings between tasks and VMs in a dynamic edge environment. Thus, LA is considered a suitable candidate for edge resource allocation problem. Researchers have proposed a lot of heuristics, meta-heuristics, machine learning, and deep learning solutions for resource allocation problems in edge environments. Some of these approaches focus on single objective [4], and some on bi-objective [4] problems. Further, some approaches need a lot of parameter updation to reach the solution [10], [11] and are time-consuming. This work differs from the existing approaches in the following ways: a Learning Automata(LA)-based solution is proposed for this bi-objective minimization problem. LA is a simple reward-penalty-based method with few mathematical operations and is not time-consuming.

The contributions made in this paper are listed below.

- First, a three-layer network architecture of an IoT-enabled smart city environment is presented for reference.
- The edge resource allocation problem to minimize energy consumption and computing delay is formulated as a bi-objective minimization problem. A Learning Automata(LA)-based edge resource allocation method is proposed for this bi-objective minimization problem.
- A series of experiments were conducted to confirm the effectiveness of the LA-based solution. The comparison results with its peers in terms of energy consumption, average delay, and success rate show the applicability of the proposed scheme in the IoT-enabled smart city environment.

The paper is organized as follows: The literature review is discussed in Section 2. Section 3 presents the three-layer network architecture of an IoT-enabled smart city. The formulation of the energy and delay minimization problem is described in Section 4. Section 5 demonstrates the fundamentals of Learning Automata (LA) and the proposed Variable Structure LA (VLA) model. The proposed LA-based edge resource allocation method, along with an example, is presented in Section 6. Section 7 and Section 8 includes the discussion of the simulation results and the concluding remarks of the paper, respectively.

2. Related work

A significant amount of work has been done in the field of edge resource allocation for IoT applications. Some of them are discussed below. The authors in [2] proposed a task offloading policy using k-means clustering and Q-learning to minimize energy consumption and delay in the IoT network. Zhao *et al.* [12] used clustering and enumeration techniques to design edge resource allocation algorithms to reduce the average response time of a request in IoT-enabled smart cities. Wang *et al.* [4] presented a hybrid genetic simulated annealing mechanism for IoT networks to minimize latency and provide energy-efficient service. Fan *et al.* [5] proposed an Application awaRE workload Allocation (AREA) algorithm to minimize response time in edge-based IoT network. A time and energy efficient resource management mechanism are

presented in [3] for IoT-fog-cloud architecture. Furthermore, a resource allocation method with an edge resource sharing contract is discussed in [6] for delay-sensitive applications. Since communication plays a key role in IoT application management, authors in [13] presented a scheme to minimize network bandwidth usage as well as the processing overhead of the central server. Authors in [10] proposed a blockchain-based Task Offloading and Resource Allocation (TORA) algorithm to address the delay and reliability issues in the IoT environment. A Software Ddefined Network (SDN) is a good candidate for implementing the IoT network, since the decision making is faster in SDN. In this context, Sahoo *et al.* [14] proposed a method to select a controller using traffic priorities and response time in a software-defined wide area network system.

Authors in [15] have addressed the issue of minimizing content download latency in edge computing with Content Caching (CC) and User Association (UA) algorithms. In [16], an algorithm is proposed to minimize the resource provisioning cost and improve resource utilization in a mobile cloud environment. Researchers used game theory and Lyapunov optimization theory-based solutions for energy harvesting in mobile edge computing [11]. The minimization of energy consumption using energy beamforming and time-based task allocation is discussed in mobile edge computing [17]. Authors in [18] have presented algorithms using the Stackelberg game concept to allocate idle resources in volunteer vehicles to address the overloaded tasks in Vehicular Edge Cloud servers. Artificial Intelligence (AI) is vital for wireless access to IoT networks. In this context, researchers in [19] presented neural networks-based and Deep Reinforcement Learning (DRL) based methods for spectrum access and spectrum sensing in centralized and distributed AI-enabled IoT networks. Correlations between the Sustainable Development Goals (SDGs) and Information and Communication Technologies (ICTs) are discussed in [20].

The Learning Automata (LA) can be used in a dynamic environment such as edge computing where the user's demand changes depending on time and location. It is used in many areas such as vertex coverage problems, VM consolidation, server allocation, etc. Some of them are discussed below. A survey on learning automata , which includes norms and behavior of learning automata is presented in [7]. The authors in [8] proposed an LA-based approach to solve a minimum vertex-covering problem in a stochastic graph. Moreover, LA is used in [9] to detect overloaded Physical Machines (PM). This result is further used to reduce energy consumption through VM consolidation. The authors in [21] have used the concept of learning automata to design an algorithm for delay-sensitive tasks to minimize energy usage and makespan in the cloud system. The behavior of learning automata with changing the number of actions is discussed in [22]. A service migration minimization method using learning automata is introduced in [23] to enable low latency, high reliability applications to have short downtime in an edge computing environment. The authors in [24] used learning automata to address the barrier coverage problem in the smart ocean IoT environment. Gheisari *et al.* [25] presented a cognitive congestion control mechanism for IoT using a learning automata game. A learning automata-based multiobjective hyper-heuristic is presented in [26]. Different types of learning automata models with their operation are discussed in [27]. Velusamy *et al.* [28] presented a learning automaton-based method for making per-request decisions to reduce the routing cost of a web request. Researchers in [29] used learning automata and autonomic computing paradigm for cloud resource provisioning to address fluctuating demand in the multiplayer online gaming system. From the above study, we found that learning automata is an interesting concept that has found applications in various research areas such as cloud, network, gaming environment, etc. However, very little work has been done on the implementation of LA for delay-constrained tasks in the edge computing environment. In this context, reinforcement-based learning and less data required for computation attract us to use LA to solve edge resource allocation problems.

3. System architecture

The three-layer network architecture of an IoT-enabled smart city is shown in Figure 1. The bottom layer or IoT layer includes IoT devices of various smart city applications such as smart homes, smart transportation, etc. However, the IoT devices have shortcomings such as limited computation capacity and low battery life. These limitations cause task offloading to the upper layer. The middle layer or edge layer and cloud layer consist of Virtual Machines (VMs). A VM in the edge layer can act as a controller. Each controller has its coverage area, and it manages IoT devices in its coverage area. Since delay constraint (i.e., deadline) is one of the key performance metrics of smart city applications [12], a controller checks whether the response time of a task meets the deadline or not. If the condition doesn't hold, then the task is rejected. Otherwise, a task offloading process is initiated.

A controller has two sub-components; decision-maker and the resource allocator (shown in Figure 3). The decision maker uses the proposed variable structure LA (VLA) model (discussed in the next section) to find an appropriate VM for a task while satisfying the system objective. The VLA model runs for a fixed number of iterations. In each iteration, the decision-maker selects a VM based on its action probability value, and its result is stored in the LA table. The

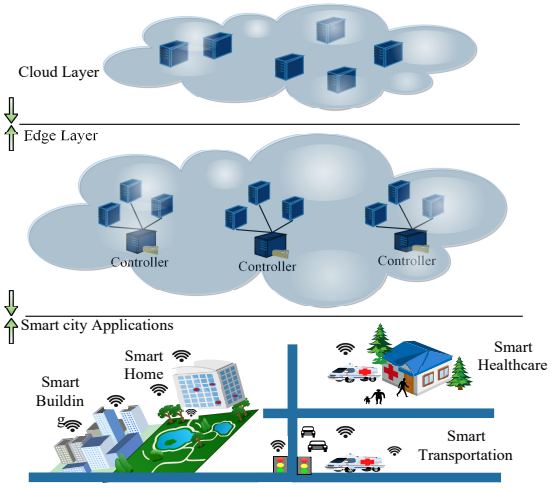


Fig. 1: Network Architecture for IoT enabled Smart City

resource allocator uses the decision of final iteration stored in the LA table to assign a task to an appropriate VM. The task with the largest computation or storage requirements are sent for processing to the topmost cloud layer.

4. Problem formulation

Suppose there are n number of IoT devices in a smart city application. The task or data from these devices are processed either locally or at the edge and cloud layer. Each task tsk_i of the IoT devices has the following characteristics: the size of the task (tsk_sz_i), tsk_en_i is the energy consumption per CPU cycle, and deadline dl_i. Let the computing capacity of the device dev_i is cc_i and it is measured as CPU cycles per second. Depending on the computing location, three modes of operation are possible: local mode, edge mode, or cloud mode. In this paper, the problem formulation is done considering local and edge modes. Suppose the edge layer has a m number of VMs. The VM characteristics are as follows: vm_cc_j is the computing capacity of vm_j is the CPU cycles per second and vm_en_j measured as energy consumption per CPU cycle. The following discussion highlights the computation delay and energy consumption in each mode.

Local Mode: The task computation is performed in the IoT device. The execution time $ext(i,l)$ of tsk_i is computed as,

$$ext(i,l) = \frac{tsk\_sz_i}{cc_i} \tag{1}$$

The start time $st(i)$ of tsk_i at device dev_i is added with the execution time $ext(i,l)$ to generate the computation delay $c\_delay(i,l)$ as computed in 2.

$$c\_delay(i,l) = st(i) + ext(i,l), \tag{2}$$

The energy $en(i,l)$ consumed by dev_i for execution of tsk_i is

$$en(i,l) = tsk\_sz_i \times tsk\_en_i \tag{3}$$

Edge or Cloud Mode: In this mode, a task tsk_i is offloaded to the edge layer. Let VM vm_j is used to process tsk_i. This task offloading causes transmission delay and is calculated as

$$tt_i = \frac{tsk\_sz_i}{cc\_link} \tag{4}$$

where cc_link is the transmission capacity of the communication link used for task offloading. The computation delay $c\_del(i,j)$ of tsk_i caused by task offloading is computed as

$$c\_del(i,j) = tt_i + (st(i,j) + ext(i,j)), \tag{5}$$

where the start time and execution time of tsk_i on vm_j are represented by $st(i,j)$.and $ext(i,j)$, respectively. The execution time is formulated as

$$ext(i,j) = \frac{tsk\_sz_i}{vm\_cc_j}. \tag{6}$$

There are two types of energy consumption associated with a task tsk_i: transmission energy $Tr\_E(i)$ due to task offloading and computation energy $C\_E(i,j)$ for task execution in a VM. Let T_E be the transmission energy consumed per CPU cycle. The transmission energy and computation energy are formulated as follows:

$$Tr\_E(i) = T\_E \times tsk\_sz_i \tag{7}$$

$$C\_E(i,j) = vm\_en_j \times tsk\_sz_i \tag{8}$$

Hence the total energy $Tot\_E(i,j)$ consumed for executing a task is

$$Tot\_E(i,j) = Tr\_E(i) + C\_E(i,j) \tag{9}$$

A binary indicator y_i is used to indicate the mode of computation. Mathematically it is represented as

$$y_i = \begin{cases} 0 & Local\ \ Mode \\ 1 & Edge\ \ Mode\ or\ Cloud\ \ Mode \end{cases} \tag{10}$$

Some tasks of an application are performed in local model and some in edge mode. So the delay $del(i,j)$ and the energy consumption $en(i,j)$ of task tsk_i are formulated as

$$del(i,j) = (1 - y_i) \times c\_delay(i,l) + y_i \times c\_del(i,j) \tag{11}$$

$$en(i,j) = (1 - y_i) \times en(i,l) + y_i \times tot\_en(i,j) \tag{12}$$

The above equation must satisfy the following constraint

$$del(i,j) \leq dl_i \quad and \quad en(i,j) \leq en\_max \tag{13}$$

Where en_max is the maximum energy consumption possible in the system. The bi-objective minimization problem is formulated as a single objective (cf_i^j) minimization problem in Equation 14. After applying the normalization process (to make both energy and delay

metrics unit less), the minimization problem is formulated as

$$cf_i^j = \alpha \times \frac{del(i,j)}{del\_max} + (1-\alpha) \times \frac{en(i,j)}{en\_max} \quad (14)$$

where α is the weight factor. The worst case delay is represented by del_max. Extending the above formulation for n number of tasks the minimization problem is reduced to

$$cf_t = \sum_{j=1}^{m} \sum_{i=1}^{n} cf_i^j \times z_i^j \quad (15)$$

where binary indicator z_i^j indicates whether tsk_i is executed on vm_j or not. The goal is to minimize cf_t. Mathematically, this is expressed as

$$Minimize \quad cf_t \quad (16)$$

5. Learning automata

Learning Automata (LA) is a type of reinforcement learning mechanism that learns by performing a finite set of actions on a random environment and receiving reinforcement signals from this environment [7],[8] [9],[22],[23],[24],[25],[26],[27][30]. It can be considered as an adaptive decision that learns to select the best action from the available set of actions and to adapt to the changes made in the working environment. The finite number of actions of an automaton is associated with probability. An action with the highest probability is selected from the problem-dependent action set. The action is performed on the environment. A reinforcement signal is generated by the environment and is sent to the automaton. Upon receiving the reinforcement signal, the automaton classifies its action as desirable or undesirable. If the action is desirable, the action taken is rewarded, otherwise the action is punished.

The automaton then updates its action probability vector based on the reinforcement signal. In this way, the automaton learns to choose the best action from its action set. The learning process continues until a stopping criterion is reached. The stopping criterion can be the maximum number of iterations or the probability value reaching a threshold. The relationship between the learning automata and its environment is shown in Figure 2, where the environment consists of virtual machines. The random environment can be of the following types, depending on the nature of the reinforcement signal: P-model, Q-model, and S-model. The P-model environment generates binary elements, success (0) or failure (1). For favorable response, the reinforcement signal value is either success or 0. An unfavorable response is indicated by failure or 1. In the Q-model environment, the reinforcement signal value is in the interval [0, 1]. In the S-model environment, the reinforcement signal value falls in the range [a,b]. Stochastic LA can be classified into finite structure LA (FLA) and variable structure LA (VLA). In VLA, the action set of an automaton varies with time, whereas in FLA action set is constant. Since the number of requests in the edge environment varies with time, we proposed a VLA model to represent it. The proposed VLA model uses P-model for reinforcement signal values.

5.1. Proposed VLA model

In this work, LA is used to find the best edge resource and IoT request pair to solve the edge resource allocation problem. We assumed that each task generated by an IoT device is associated with an automaton and the VMs present in the edge layer realize the random environment. The proposed VLA model, as shown in Figure 3, is represented by the tuple $< tsk_i, A_i, \alpha_i, p_i, \beta_i, L >$. tsk_i is the task generated by i^{th} IoT device, A_i is the learning automaton of tsk_i, action set of A_i is α_i, the probability vector corresponding to α_i is p_i, the reinforcement signal for action α_i is β_i and L is the learning algorithm. Here, the mapping of a task to a VM is considered as an action of an automaton. Thus, α_i can be formulated as $\alpha_i = \{\alpha_i^j | 1 \leq j \leq m, i = 1\ to\ n\}$, where α_i^1 denotes the mapping of task tsk_1 to VM vm_1, α_i^2 means tsk_i is mapped to vm_2 and so on. Similarly, the probability vector $p_i = \{p_i^j | \alpha_i^j \in \alpha_i\}$, where p_i^1 means the probability associated with action α_i^1, p_i^2 means the probability associated with action α_i^2 and so on. Since the action with the highest probability is chosen by an automaton at every moment or iteration, it can be rewritten $p_i = max\{p_i^1, p_i^2, ..., p_i^m\}$ and $\alpha_i = \{\alpha_i^j | p_i^j = max(p_i)\}$. The learning algorithm L is a

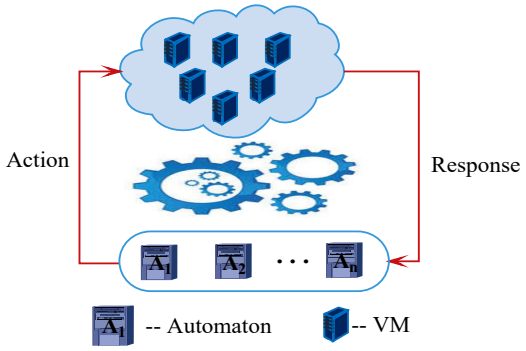


Fig. 2: Learning automata and its Environment

recurrence relation adopted to revise the action probability vector. The learning algorithm uses the cost function value at iteration $k-1$ and k to generate the reinforcement signal. The action α_i of A_i is either rewarded or penalized according to the β_i value. This reward and penalty operation updates the action probabilities. If $\beta_i = 0$, the action of A_i is desirable, then the action is rewarded using Equation 17. If $\beta_i = 1$, then penalize the action of A_i according to the Equa-

tion 18.

$$p_i^j(k+1) = \begin{cases} p_i^j(k) + \phi \times (1 - p_i^j(k)) & j = i \\ (1-\phi) \times p_i^j(k) & \forall j, j \neq i \end{cases} \quad (17)$$

$$p_i^j(k+1) = \begin{cases} (1-\varphi) \times p_i^j(k) & j = i \\ \frac{\varphi}{r-1} + (1-\varphi) \times p_i^j(k) & \forall j, j \neq i \end{cases} \quad (18)$$

Here, ϕ and φ are the reward and penalty constants, respectively. For the following iteration, Equation 17 says to increase the probability of action α_i at iteration k (first part) and decrease the probability of other actions (second part). Similarly, Equation 18 says that, for the next iteration, p_i^j of α_i^j is decreased (first part) while p_i^j, $j \neq i$, of other actions are increased (second part). The LA table stores the result of each iteration. The proposed VLA model is designed for a given task set, but it can be extended to support other task sets. This feature of the proposed VLA makes it adaptable to the changing needs of in IoT-enabled smart city environments.

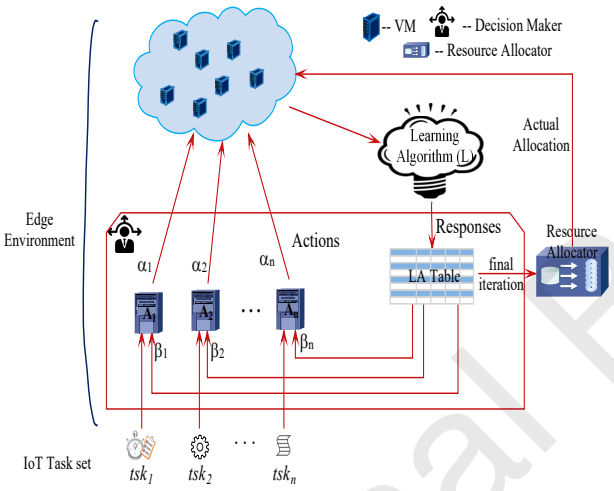


Fig. 3: Proposed VLA Model

6. Learning automata based edge resource allocation approach

For a dynamic edge environment, a fast decision unit is required to help satisfy the delay constraint of tasks. The unit will be responsible for making resource allocation decisions based on the previous decision in a dynamically changing environment. This gives us the motivation to apply the learning method to the edge resource allocation problem. In the proposed learning automata (LA) based approach, the continuous interaction between the automaton and the environment leads to the best decision. The unique features of LA that justify its choice for the edge resource allocation problem are listed below.

- LA requires no a priori information of an underlying application and has a simple feedback system.
- LA has very few mathematical operations, and it doesn't need to perform time-consuming calculations, which makes it suitable for delay-sensitive applications.
- The responses of the environment to the automaton are reflected in the action probability matrix. Tuning this history information can help to improve system performance.

Algorithm 1 : Pseudo code of LA-based Edge Resource Allocation Approach

Input: Task set (Tk), $tsk_i \in Tk$,VM set Vm, $vm_j \in Vm$
Output: Scheduled pair $\langle tsk_i, vm_j \rangle$

Set $it = 1$;
for each (tsk_i, vm_j) pair in the system **do**
    Generate execution time ET matrix and probability matrix P using $Step-1$ and $Step-2$;
end for
while $it \leq it\_max$ **do**
    for each (tsk_i, vm_j) pair in the system **do**
        if $it == 1$ **then**
            Randomly assign a task to a VM;
        else
            Choose an action with highest probability in P;
        end if
    end for
    Compute the cost function for the assignment using Equation 15.
    if $cf_t(it) \leq cf_t(it-1)$ **then**
        for each (tsk_i, vm_j) pair in the system **do**
            if $del_i^j \leq dl_i$ **then**
                Set $\beta_i = 0$;
            else
                Set $\beta_i = 1$;
            end if
        end for
    end if
    for each task tsk_i in the system **do**
        if $\beta_i == 0$ **then**
            Use Equation 17 to reward action of A_i;
        else if $\beta_i == 1$ **then**
            Use Equation 18 to penalize action of A_i;
        end if
    end for
    Update p_i^j value using $Update\_Prob()$;
    $it = it + 1$;
end while

The steps followed in the proposed LA-based approach are discussed below.

- *Step-1-Initialization:* Initially, all VMs are available in the edge layer. So, the probability p_i^j of

each action α_i^j of the automaton A_i is set to $\frac{1}{m}$, where m is the number of VMs in the edge layer. Thus, $p_i^j(0) = \frac{1}{m}$. Also, $cf_t(0) = INF$.

- *Step-2-Representation of Execution and Probability Matrix:* The set of actions α_i of an automaton A_i is the i^{th} row element of the execution matrix ET. For example, actions α_1^j, $j = \{1, 2, ..., m\}$ for automaton A_1 are the row 1 elements of ET. Similarly, p_i^j is the element of $n \times m$ action probability matrix P.

- *Step-3-Update Action Probability:* The learning algorithm L decides whether an action α_i^j by A_i is rewarded or punished. Let θ_min denote the minimum value and θ_max indicates maximum value for probability p_i^j. If $p_i^j \leq \theta\_min$ for an action then deactivate it for task tsk_i. Similarly, if $p_i^j = \theta\_max$, then deactivate all other actions and assign tsk_i to VM vm_j. The deactivated actions are removed from the action set of an automaton. This process is done to avoid unnecessary computation, which contributes to faster computation. The pseudocode for updating the action probability is shown in Algorithm 2.

- *Step-4-Stopping Criteria:* If iteration it equals maximum iteration count it_max, stop the learning process.

The operation of the proposed LA-based edge resource allocation mechanism is shown in Algorithm 1. Initially, the iteration count it is set to 1. For each $(task_i, vm_j)$ pair in the system the execution time ET and the probability matrix are computed (line - 3). For the first iteration, a task is randomly assigned to an available VM. However, in a subsequent iteration, an action with the highest probability is chosen for the assignment (lines 7 - 11). In each iteration, the cost function is computed for the resource allocation decision made. The cost function value in the current iteration (say, it) is compared to the previously computed (i.e., $it - 1$) cost function value. If $cf_t(it) \leq cf_t(it - 1)$, then it is checked whether the action of the automaton satisfies deadline constraint or does not. If the condition holds, then set reinforcement signal $\beta_i = 0$, otherwise set $\beta_i = 1$ (lines 13 - 22). This reinforcement signal value is used to reward or penalize an action (lines 24 - 28). Accordingly, the action probability value is updated. This process continues until $it \leq it\_max$ is not reached (lines 30 - 32).

Algorithm 2 : Pseudo code of $Update\_Prob()$

Input: Task set (Tk), $tsk_i \in Tk$,VM set Vm, $vm_j \in Vm$, θ_min, α_i^j, θ_max

Output: Decision on Action α_i^j

1: **for** each (tsk_i, vm_j) pair in the system **do**
2: **if** $p_i^j \leq \theta\_min$ **then**
3: Deactivate α_i^j;
4: **end if**
5: **if** $p_i^j == \theta\_max$ **then**
6: Assign tsk_i to vm_j;
7: **end if**
8: **end for**

6.1. Example

The Algorithm 1 is explained with an example. Let a smart city application has three IoT devices ($n = 3$), and the edge layer has two VMs ($m = 2$). Then, the list of automatons is A_1, A_2, A_3 (one for each IoT device). An automaton A_i, $i = \{1, 2, 3\}$ has two actions ($r = 2$). The two actions are the task generated by A_i can be processed either on vm_1 or at vm_2. So, the actions of automaton A_i are represented by α_i^j, $i = \{1, 2, 3\}$ and $j = \{1, 2\}$. Assume that the maximum number of iterations is $it\_max = 3$.

Iteration 1 : Let the action set $\alpha = \{\alpha_1^2, \alpha_2^1, \alpha_3^1\}$, i.e., tsk_1 associated with A_1 is assigned to vm_2 and so on. This scenario is shown in Figure 4.

Suppose α_1^2 and α_2^1 meet the deadline constraint of respective tasks. Whereas α_3^1 fails to meet the deadline constraint of the task. Assume that the cost function value for above edge resource allocation is $cf_t(1) = 18$. Since, $cf_t(1) \leq cf_t(0)$, the reinforcement signal generated by the environment is $\{\beta_1 = 0, \beta_2 = 0, \beta_3 = 1\}$. Based on the β_1 and β_2 value, actions α_1^2 and α_2^1 are rewarded. The action α_3^1 is penalized as $\beta_3 = 1$. Let $\phi = \varphi = 0.1$. The detailed calculation (reward) of A_1 and A_2 using Equation 17 are as follows:

$$\begin{cases} p_1^1(2) = (1 - \phi) \times p_1^1(1) = 0.9 \times 0.5 = 0.45 \\ p_1^2(2) = 0.5 + 0.1 \times 0.5 = 0.55 \\ p_2^1(2) = (1 - \phi) \times p_2^1(1) = 0.5 + 0.1 \times 0.5 = 0.55 \\ p_2^2(2) = 0.9 \times 0.5 = 0.45 \end{cases} \tag{19}$$

Similarly, the penalty calculation of A_3 using Equation 18 is as follows:

$P =$

Task \VM	vm_1	vm_2
tsk_1	0.5	0.5
tsk_2	0.5	0.5
tsk_3	0.5	0.5

Allocation=

tsk_1	tsk_2	tsk_3
vm_2	vm_1	vm_1

Fig. 4: Probability Matrix and Allocation Vector (Initial)

$$\begin{cases} p_3^1(2) = 0.9 \times p_3^1(1) = 0.9 \times 0.5 = 0.45 \\ p_3^2(2) = 0.1 + 0.9 \times p_3^2(1) = 0.1 + 0.9 \times 0.5 = 0.55 \end{cases} \tag{20}$$

$P =$

Task \VM	vm_1	vm_2
tsk_1	0.45	**0.55**
tsk_2	**0.55**	0.45
tsk_3	0.45	**0.55**

Allocation=

tsk_1	tsk_2	tsk_3
vm_2	vm_1	vm_2

After Iteration 1

Fig. 5: Probability Matrix and Allocation Vector (Iteration 1)

The updated probability matrix after iteration 1 is shown in Figure 5.

Iteration 2 : Let the action set is $\alpha = \{\alpha_1^2, \alpha_2^1, \alpha_3^2\}$, and all the actions meet the deadline limit of the respective tasks. The reinforcement signal of each action is set to zero,i.e., $\{\beta_1 = 0, \beta_2 = 0, \beta_3 = 0\}$. The probability matrix is updated as follows:

$P =$

Task \VM	vm_1	vm_2
tsk_1	0.405	**0.595**
tsk_2	**0.595**	0.405
tsk_3	0.405	**0.595**

Allocation=

tsk_1	tsk_2	tsk_3
vm_2	vm_1	vm_2

Fig. 6: Probability Matrix and Allocation Vector (Iteration 2)

$$\begin{cases} p_1^1(2) = 0.9 \times 0.45 = 0.405 \\ p_1^2(2) = 0.55 + 0.1 \times 0.45 = 0.595 \\ p_2^1(2) = 0.55 + 0.1 \times 0.45 = 0.595 \\ p_2^2(2) = 0.9 \times 0.45 = 0.405 \\ p_3^1(2) = 0.9 \times 0.45 = 0.405 \\ p_3^2(2) = 0.1 + 0.9 \times 0.55 = 0.595 \end{cases} \quad (21)$$

The updated probability matrix after iteration 2 is shown in Figure 6. This is the final edge resource allocation decision.

6.2. Time complexity

Theorem 1. *The time complexity of the proposed edge resource allocation algorithm is* $O(nm)$.

Proof. To generate ET and P matrix $O(nm)$ time is required. The time complexity to generate the allocation vector is $O(nm)$. The cost function calculation also takes $O(nm)$ times. $O(nm)$ time is needed to check whether a task meets it's deadline or not. The time complexity of computing the reward and penalty value is $O(n)$. The time required to set the reinforcement signal value is $O(nm)$. The time complexity of updating the probability matrix is $O(nm)$. Thus, the time complexity of the proposed LA-based edge resource allocation approach is $O(nm) + it\_max(O(nm) + O(nm) + O(nm) + O(n) + O(nm) + O(nm)) = O(nm)$. □

7. Simulation results

The following algorithms are used for comparison: local processing, edge offloading, and EOERA [12] to show the performance improvement of the proposed LA-based edge resource allocation algorithm. The details of the algorithms are as follows:

- Local processing:- In this approach, computation is done at the IoT devices.
- Edge offloading:- The tasks whose computation cannot be completed by local processing are sent to the edge layer. A greedy approach is used in the edge layer to execute a task.
- Enumeration-based Optimal Edge Resource Allocation (EOERA): All the possible edge resource allocations are enumerated, and the response time is calculated for each case. All the calculated values are compared, and an optimal allocation is selected. This work has similarities with our work in the sense that "in both cases, all possible edge resource allocation combinations are considered and compared to find the optimal solution". The difference between this work [12] and our work is that the edge resource allocation in [12] is a single objective problem (latency minimization). Whereas our approach is a bi-objective (energy consumption and delay minimization) edge resource allocation problem. Second, our approach assigns a probability value to each (task, VM) pair, which helps to reduce the search time as the pair with the highest probability is always selected. In contrast, [12] searches the entire solution space is searched to find the optimal solution.

Table 1: Simulation Settings

Parameter	**Value**
tsk_i size	[10 – 30] *Mcycles* [4]
Energy consumed by dev_i	10^{-7} *Joules per cycle* [2]
dev_i's computation capacity	[150 – 600] *Mcycles per second* [4]
Energy consumed by vm_j	10^{-8} *Joules per cycle* [2]
vm_j's computation capacity	$[10^4 - 3 \times 10^4]$ *Mcycles per second* [4]
Computation capacity of transmission link	$[10^2 - 2 \times 10^2]$ *Mcycles per second*
Transmission energy	10^{-7} *Joules per cycle*

The performance of the system is evaluated in terms of energy consumption, average delay, and success rate. The number of tasks completed before the deadline divided by the total number of tasks is called the success ratio. The performance of the algorithms is evaluated by varying the number of tasks and the number of VMs. The experiment is carried out using an in-house simulator and Python 3.0 on an Intel (R) Core (TM) i7-4770 CPU @ 3.40 GHz 3.40 GHz CPU and 8 GB RAM running on WINDOWS 11. The detailed simulation setting and parameter is given in TABLE 1. The simulation framework consists of a task generator, task scheduler, and VM pool. A task generator is designed to generate tasks where task arrival follows the Poisson distribution. We assume that the task arrival rate is $\lambda = 0.8$. The task's deadline is set as: $dl_i = a_i + baseD$, where $baseD$ is in the uniform distribution $U(5, 10)$. VM pool consists of a finite set of VMs and represents a cloud environment for task

scheduling. Task scheduler uses scheduling mechanisms (algorithms) and various constraints to map a task to an appropriate VM in the VM pool.

The reward (ϕ) and penalty (φ) values determine the speed and accuracy of convergence of the LA approach. If ϕ is too small, the learning process will be slow. Whereas, if ϕ is too large, the action probability value will be large and the accuracy of the automaton in perceiving the optimal behavior will be low. Thus, an appropriate ϕ value that is sufficiently small can cause the probability of convergence to the optimal behavior as close to 1 as desired. Furthermore, to measure the learning ability of LA, it is compared to a pure chance automaton that always chooses its actions with equal probabilities. We conduct an experiment to choose an appropriate value for ϕ and φ. An experiment is carried out with a task count between [50 − 300] with a step of 50. The VM count is set to 60. The variation of the cost function value with the proposed LA-based approach and the pure-chance (all action probabilities are equal) approach over the task count is shown in Figure 7a with $\phi = \varphi = 0.1$, Figure 7b with $\phi = \varphi = 0.01$, Figure 7c with $\phi = \varphi = 0.001$, respectively. From Figure 7a to Figure 7c, it can be inferred that the value of the cost function increases with an increase in the number of tasks, even when the values of ϕ and φ change. Further, it can be seen that a low value for $\phi = \varphi = 0.001$ causes a rise in the cost function value. This also indicates the slow learning process of LA. Here, we consider $\phi = \varphi = 0.1$. It can be seen from Figure 10 that, after 500 iterations there is minimal variation in the objective function value. This indicates automata is starting to converge to a solution. So, the maximum iteration count *it_max* is set to 500.

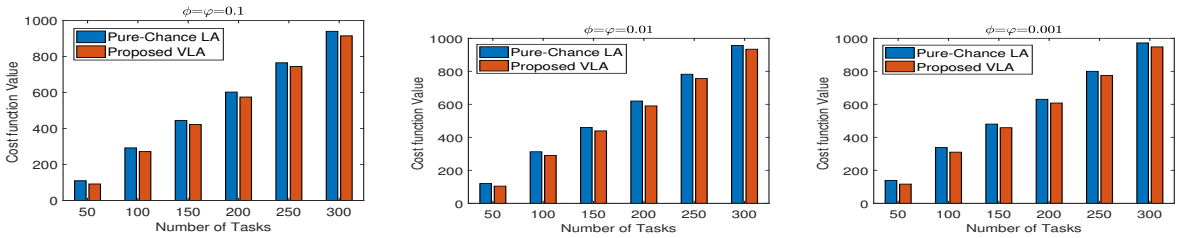
Figure 8 shows the effect of the number of tasks on system performance. We set the number of VMs to 60. The average computation delay increases as the number of tasks increases,as shown in Figure 8a. The proposed LA-based method has a shorter delay compared to local processing and edge offloading. This can be attributed to the fact that the computing capability of IoT devices is low, which makes it difficult to handle too many tasks. Whereas task offloading to the edge layer has low delay but has a bottleneck of transmission delay. The fixed number of VMs with an increase in the number of tasks inevitably an increase in the computation delay. However, the LA-based resource allocation approach saves the computing delay significantly. The energy consumption increases as the number of tasks increases, as shown in Figure 8b. The device consumes a lot of energy while processing a task. The increase in task count keeps the VM busy, which in turn consumes computation energy. In addition, the idle time of VM also contributes to the energy consumption. There is also transmission energy that adds to the total energy consumption. The experimental result as shown in Figure 8b indicates that the LA concept helps the proposed to efficiently utilize the resource (VM), which in turn saves more energy. The effect of the number of tasks on the success rate is shown in Figure 8c. Local processing will not have a significant success rate due to the limited computing capability of the IoT device. So, local processing is not considered for comparison. Adding tasks with a fixed number of VMs causes many tasks to miss their deadlines. This reduces the success rate. However, the proposed approach has a high success rate compared to all the other approaches. The VM selection based on the probability value helps the proposed LA-based approach to select the best VM for a task.

	Experiment	Edge offloading	EOERA	Proposed Method		
	1	77	82	84		
	2	83	78	89		
	3	85	87	93		
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Rows	84.22	2	42.11	5.3	0.075	6.94

Fig. 11: ANOVA Test Scenario

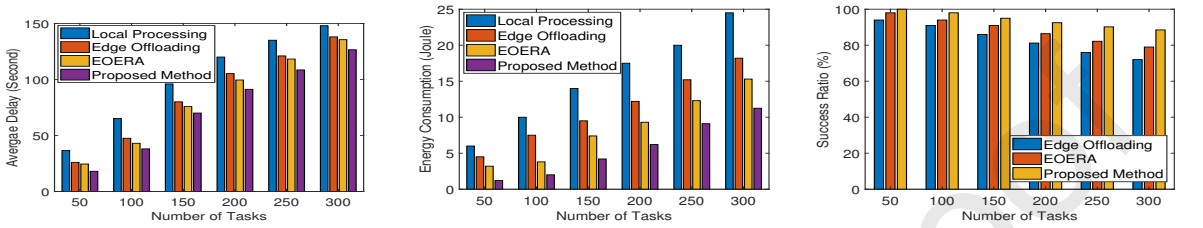
The effect of the number of VMs on system performance is shown in Figure 9. Here, number of tasks is set to 200. Figure 9a shows the behavior of the system as the number of VMs changes. Adding VMs to the system reduces the computing delay. This is because a large number of VMs allows many task to be executed in parallel. The transmission delay can be ignored due to its small value, as computing delay is the main contributor to the total delay. However, the edge offloading approach has more computation delay compared to the proposed LA-based method. The continuous interaction of LA with the working environment to select the best VM for a task causes the performance improvement of the proposed method. The energy consumption with variation in the number of VMs is shown in Figure 9b. The addition in VM count increases the number of successful task executions. This contributes to the computation energy. The transmission energy also contributes to total energy consumption. In addition, the total energy consumption also increases due to the energy consumed by idle VM in a large VM count. The edge offloading process consumes more energy compared to the proposed LA-based method. The decision on the current VM selection considering the previous selection in the proposed LA-based method guarantees this performance improvement. In Figure 9c shows the effect of the number of VMs on the success rate. By increasing the number of VMs, more tasks can be executed and completed before the specified deadline. Thus, the success rate increases as the number of VMs increases. From the figure, it can be observed that the proposed approach has a better performance compared to others. This is because the continuous interaction with the environment and the probability value allows the selection of the best (task, VM) pair.

A two-way Analysis of Variance i.e. ANOVA test is performed with the null hypothesis: "All algorithms



(a) Cost function Value with $\phi = \varphi = 0.1$ (b) Cost function Value with $\phi = \varphi = 0.01$ (c) Cost function Value with $\phi = \varphi = 0.001$

Fig. 7: Cost function value for different ϕ and φ value



(a) Average Delay (b) Energy consumption (c) Success Ratio

Fig. 8: Impact of number of tasks on system performance

have a success rate of 70%". The test was applied to 350 tasks with different numbers of VMs; 30 (Experiment 1), 60 (Experiment 2), and 90 (Experiment 3), as shown in shown in Figure 11. The calculated p-value for the rows is 0.075, which is greater than the default value of 0.05. Hence, the null hypothesis cannot be rejected at the 95% level of confidence level.

8. Conclusion

Today, IoT applications are present everywhere starting from the smart grids to the smart healthcare systems. These applications require low computing latency. Energy consumption is another important issue in the IoT network. In this regard, edge computing is helpful because it brings computation close to the data source, thus reducing computation delay and energy consumption. The energy and delay minimization problem addressed in this work has been formulated as a bi-objective edge resource allocation problem. First, a three-layer IoT network architecture for IoT-enabled smart cities was presented. Then learning automata (LA) had been used to realize the solution of the above problem. First, an LA-based edge resource allocation framework was designed, and then an allocation algorithm was presented considering this framework. The simulation results show the performance improvement of the proposed LA-based edge resource allocation method over some existing approaches. In the future, we plan to implement this approach in a specific smart application, such as a smart home or smart driving. In addition, we plan to develop an edge resource allocation method that considers other metrics such as task allocation fairness and user satisfaction.

Acknowledgement

This paper was supported by the Kempe postdoc fellowship via Project No. SMK21-0061, Sweden. Additional support was provided by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

References

[1] S. Sahoo, K. S. Sahoo, B. Sahoo, A. H. Gandomi, An auction based edge resource allocation mechanism for iot-enabled smart cities, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2020, pp. 1280–1286.

[2] X. Liu, J. Yu, J. Wang, Y. Gao, Resource allocation with edge computing in iot networks via machine learning, IEEE Internet of Things Journal 7 (4) (2020) 3415–3426.

[3] H. Sun, H. Yu, G. Fan, L. Chen, Energy and time efficient task offloading and resource allocation on the generic iot-fog-cloud architecture, Peer-to-Peer Networking and Applications 13 (2) (2020) 548–563.

[4] Q. Wang, S. Chen, Latency-minimum offloading decision and resource allocation for fog-enabled internet of things networks, Transactions on Emerging Telecommunications Technologies (2020) e3880.

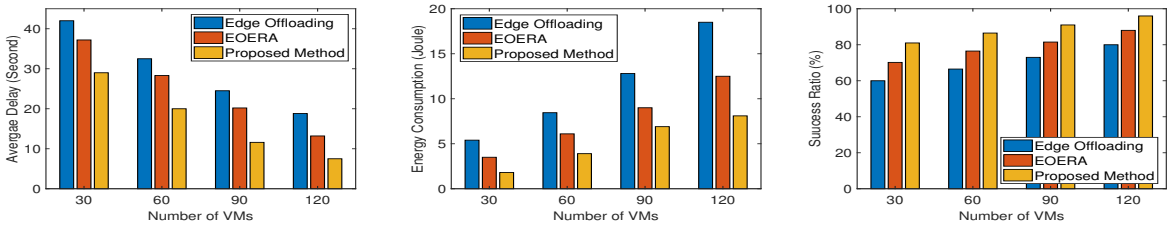
[5] Q. Fan, N. Ansari, Application aware workload allocation for edge computing-based iot, IEEE Internet of Things Journal 5 (3) (2018) 2146–2153.

[6] J. Xu, B. Palanisamy, H. Ludwig, Q. Wang, Zenith: Utility-aware resource allocation for edge computing, in: 2017 IEEE International Conference on Edge Computing (EDGE), 2017, pp. 47–54.

[7] K. S. Narendra, M. A. Thathachar, Learning automata-a survey, IEEE Transactions on systems, man, and cybernetics (4) (1974) 323–334.

[8] A. Rezvanian, M. R. Meybodi, Finding minimum vertex covering in stochastic graphs: a learning automata approach, Cybernetics and Systems 46 (8) (2015) 698–727.

[9] M. Ranjbari, J. A. Torkestani, A learning automata-based algorithm for energy and sla efficient consolidation of virtual



(a) Average Delay (b) Energy consumption (c) Success Ratio

Fig. 9: Impact of number of VMs on system performance

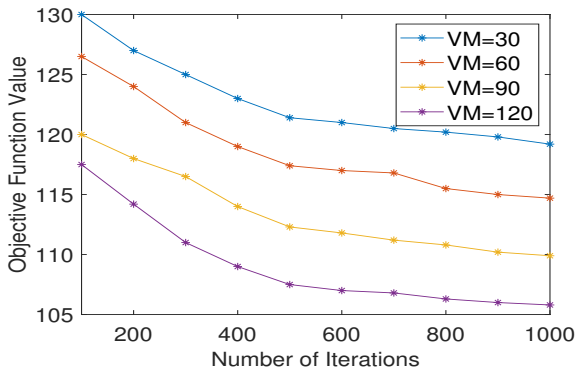


Fig. 10: Number of iteration Vs Objective Function Value

machines in cloud data centers, Journal of Parallel and Distributed Computing 113 (2018) 55–62.

[10] K. Xiao, Z. Gao, W. Shi, X. Qiu, Y. Yang, L. Rui, Edgeabc: An architecture for task offloading and resource allocation in the internet of things, Future Generation Computer Systems 107 (2020) 498–508.

[11] S. Xia, Z. Yao, Y. Li, S. Mao, Online distributed offloading and computing resource management with energy harvesting for heterogeneous mec-enabled iot, IEEE Transactions on Wireless Communications 20 (10) (2021) 6743–6757.

[12] L. Zhao, J. Wang, J. Liu, N. Kato, Optimal edge resource allocation in iot-based smart cities, IEEE Network 33 (2) (2019) 30–35.

[13] B. Mohebali, A. Tahmassebi, A. H. Gandomi, A. Meyer-Baese, A big data inspired preprocessing scheme for bandwidth use optimization in smart cities applications using raspberry pi, in: Big Data: Learning, Analytics, and Applications, Vol. 10989, International Society for Optics and Photonics, 2019, p. 1098902.

[14] K. S. Sahoo, P. Mishra, M. Tiwary, S. Ramasubbareddy, B. Balusamy, A. H. Gandomi, Improving end-users utility in software-defined wide area network systems, IEEE Transactions on Network and Service Management 17 (2) (2019) 696–707.

[15] Y. Li, H. Ma, L. Wang, S. Mao, G. Wang, Optimized content caching and user association for edge computing in densely deployed heterogeneous networks, IEEE Transactions on Mobile Computing (2020).

[16] Y. Li, J. Liu, B. Cao, C. Wang, Joint optimization of radio and virtual machine resources with uncertain user demands in mobile cloud computing, IEEE Transactions on Multimedia 20 (9) (2018) 2427–2438.

[17] S. Mao, J. Wu, L. Liu, D. Lan, A. Taherkordi, Energy-efficient cooperative communication and computation for wireless powered mobile-edge computing, IEEE Systems Journal (2020).

[18] F. Zeng, Q. Chen, L. Meng, J. Wu, Volunteer assisted collaborative offloading and resource allocation in vehicular edge computing, IEEE Transactions on Intelligent Transportation Systems 22 (6) (2020) 3247–3257.

[19] H. Song, J. Bai, Y. Yi, J. Wu, L. Liu, Artificial intelligence enabled internet of things: Network architecture and spectrum access, IEEE Computational Intelligence Magazine 15 (1) (2020) 44–51.

[20] J. Wu, S. Guo, H. Huang, W. Liu, Y. Xiang, Information and communications technologies for sustainable development goals: state-of-the-art, needs and perspectives, IEEE Communications Surveys & Tutorials 20 (3) (2018) 2389–2406.

[21] S. Sahoo, B. Sahoo, A. K. Turuk, A learning automata-based scheduling for deadline sensitive task in the cloud, IEEE Transactions on Services Computing (2019) 1–1.

[22] M. Thathachar, B. R. Harita, Learning automata with changing number of actions, IEEE transactions on systems, man, and cybernetics 17 (6) (1987) 1095–1100.

[23] A. Mukhopadhyay, M. Ruffini, Learning automata for multi-access edge computing server allocation with minimal service migration, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.

[24] X. Deng, Y. Jiang, L. T. Yang, L. Yi, J. Chen, Y. Liu, X. Li, Learning-automata-based confident information coverage barriers for smart ocean internet of things, IEEE Internet of Things Journal 7 (10) (2020) 9919–9929.

[25] S. Gheisari, E. Tahavori, Cccla: A cognitive approach for congestion control in internet of things using a game of learning automata, Computer Communications 147 (2019) 40–49.

[26] W. Li, E. Özcan, R. John, A learning automata-based multi-objective hyper-heuristic, IEEE Transactions on Evolutionary Computation 23 (1) (2017) 59–73.

[27] A. Rezvanian, B. Moradabadi, M. Ghavipour, M. M. D. Khomami, M. R. Meybodi, Introduction to learning automata models, in: Learning Automata Approach for Social Networks, Springer, 2019, pp. 1–49.

[28] G. Velusamy, R. Lent, Dynamic cost-aware routing of web requests, Future internet 10 (7) (2018) 57.

[29] M. S. Aslanpour, M. Ghobaei-Arani, M. Heydari, N. Mahmoudi, Larpa: A learning automata-based resource provisioning approach for massively multiplayer online games in cloud environments, International Journal of Communication Systems 32 (14) (2019) e4090.

[30] S. Sahoo, B. Sahoo, A. K. Turuk, An energy-efficient scheduling framework for cloud using learning automata, in: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2018, pp. 1–5.

Conflict of interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.