IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 14, 2021, accepted June 27, 2021, date of publication July 5, 2021, date of current version July 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094528

Time-Constrained Task Allocation and Worker Routing in Mobile Crowd-Sensing Using a Decomposition Technique and Deep Q-Learning

SHATHEE AKTER[1], THI-NGA DAO[2], AND SEOKHOON YOON[1], (Member, IEEE)
[1]Department of Electrical, Electronic and Computer Engineering, University of Ulsan, Ulsan 44610, South Korea
[2]Faculty of Radio Engineering, Le Quy Don Technical University, Hanoi 10000, Vietnam

Corresponding author: Seokhoon Yoon (seokhoonyoon@ulsan.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT under Grant 2019R1F1A1058147, and in part by the Ministry of Education under Grant 2021R1I1A3051364.

ABSTRACT Mobile crowd-sensing (MCS) is a data collection paradigm, which recruits mobile users with smart devices to perform sensing tasks on a city-wide scale. In MCS, a key challenge is task allocation, especially when MCS applications are time-sensitive, and the platform needs to consider task completion order (since a worker may perform multiple tasks and different task completion orders lead to different travel costs and response times, i.e., the times needed to arrive at the task venues), requirements of tasks (such as deadline and required sensor) and workers heterogeneity. In other words, the task allocation problem consists of multiple task completion order problems, which is challenging to solve due to the large solution space. Therefore, in this paper, we first formulate the considered problem into two related integer linear programming problems (i.e., assignment and task completion order problems) using a decomposition technique in order to reduce the problem size and enable the use of diverse searching strategies. Then, a deep Q-learning (DQN)-based algorithm, namely assignment DQN with a local search (A-DQN w/ LS), is proposed to determine the task–worker assignments, which iteratively employs an asymmetric traveling salesman (ATSP) heuristic to find the task completion orders of the workers. The local optimizer is applied at the end of the A-DQN algorithm to deal with the computation time and local optima. Simulation results show that the proposed method outperforms existing approaches under different sensing dynamics in terms of total cost.

INDEX TERMS Deep reinforcement learning, mobile crowd-sensing, task allocation, tabu search.

I. INTRODUCTION

The recent advancement and proliferation in communication technologies and sensor-equipped portable smart devices have enabled a noble data collection paradigm called mobile crowd-sensing (MCS). MCS exploits the mobility of the mobile users and their sensor-equipped smart devices (such as smartphones or smartwatches) [1] to collect and share dynamic real-time sensing information (e.g., information about traffic, drainage, and road system during heavy rain, noise quality, air pollution level, ambient context, or local information) from their surroundings through the existing communication networks (e.g., 3G, 4G, or 5G networks) [2]. Thus, it can achieve a remarkable level of spatial and temporal coverage in the region of interest without significant infrastructure and maintenance costs compared to traditional static sensor networks. Due to these advantages of MCS, it has been applied to diverse real-world applications such as crowd counting [3], environmental monitoring [4], information mapping [5], smart city [6], and intelligent transportation systems [7].

In a typical MCS system, there are three main components: the task requesters, the mobile workers or users, and the MCS platform. The task requesters publish tasks of interest to the MCS platform. The users (who are registered on the platform) perform sensing tasks through smart devices consciously (i.e., workers intentionally perform the sensing tasks) or unconsciously (i.e., sensing tasks are performed by

The associate editor coordinating the review of this manuscript and approving it for publication was Zhipeng Cai.

the smart devices without worker's knowledge) in return for some incentive, such as a monetary benefit or certain forms of entertainment [8]. The service platform assigns sensing tasks published by the requesters to suitable users or workers according to the requirements of the applications and the available MCS resources. This task assignment process is one of the key issues in MCS systems due to its significant effect on the success of the MCS applications and the efficiency of the platform.

Recently, a lot of studies have been done on MCS task allocation [9]–[17]. Earlier studies (e.g., [9]–[11]) mostly consider single-task allocation, i.e., one task is assigned to a worker or a group of workers in each round of allocation, and assign tasks by estimating the probability of workers visiting the task location during their daily routines, which is resource-inefficient and impractical when tasks are required to be completed within a deadline. Thus, in order to make full use of the limited resources and to accommodate deadline-sensitive MCS applications, recent studies consider multi-task allocation, where a worker can perform multiple tasks and needs to intentionally visit the venues to perform them (e.g., [12]–[17]).

Assigning multiple tasks to a worker creates a new challenge for the platform, i.e., finding the task completion order because different task completion orders or moving paths of the workers lead to different traveling costs and response times (defined as the times needed to arrive at the tasks' locations). Specifically, the task assignment problem is composed of several task completion orders or path-planning problems. Therefore, the search space is very large, which makes finding a good solution very difficult. However, this has rarely been considered by existing works. In addition, most of the existing works assume that the sensing duration of the tasks is negligible, which may result in a failure of service if tasks have non-negligible sensing duration.

Participations of different types of workers, such as the unmanned aerial vehicle (UAV) and human, also needs to be taken into account, where tasks can request different types of workers according to their preferences, incentives, and location accessibility. The location accessibility of the tasks means some task venues may be unsafe or risky for a human worker. For instance, collecting information about drainage systems and gas lines after a natural disaster (such as an earthquake or flood) or monitoring waste removal sites to identify hazardous materials.

Therefore, in this paper, a multi-task allocation problem is considered under the following settings: tasks are diverse (i.e., they may have different sensing durations, deadlines, and required sensor types), and a task can be performed by different types of workers (i.e., UAV and human workers). The task completion order of a worker is a Hamiltonian cycle, and finding the lowest-cost task completion order of a worker is considered a traveling salesman problem (TSP).

One simple example for a better understanding of the proposed problem is given below. Assume that eight tasks (i.e., $\{t1, t2, .., t8\}$) are requested concurrently, and each of them is associated with a deadline and task duration, as shown in Fig. 1. For instance, $t2(19, 5)$ means $t2$ has a deadline of 19 units of time, and the sensing duration is five units. Registered workers (i.e., $w1, \ldots, w4$) are recruited to perform the tasks by visiting the venues. The number associated with the worker is the demanded per-unit incentive with respect to traveling distance. Moreover, task $t4$ requests a UAV worker to perform the task due to preference or location accessibility. Consider tasks $t8$, $t1$, and $t4$ where $w2$ is the nearest worker. If $w2$ greedily selects $t8$ because it is closer than $t1$, then a feasible task sequence for $w2$ is $(t8, t1)$ according to traveling time and sensing duration, leaving $t4$ without a worker ($t4$ cannot be assigned to other workers as it has requested for the UAV). On the other hand, if $w4$ is assigned to tasks $t8$, $w2$ can be mapped to task sequence $(t1, t4)$. However, if $w2$ performs $t4$ first, the valid time of performing $t1$ will expire before the worker's arrival due to the sensing duration of $t4$, resulting in an infeasible solution.

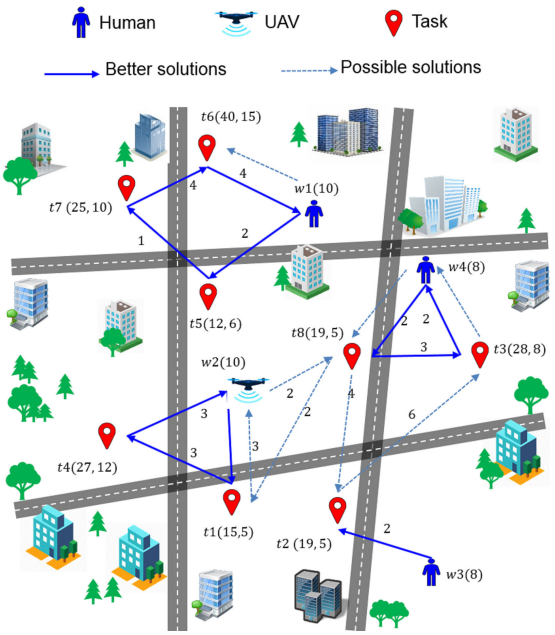

FIGURE 1. An exemplary scenario of multi-task allocation. The dotted blue lines show the possible task–worker assignments, whereas the solid blue line segments show better task–worker assignments. The numbers on the lines show the time needed for a worker to reach the task venue.

As seen from the example, there are several possible combinations of task–worker assignments and a lot of possible orders of completing tasks assigned to a worker, which exponentially increases as the number of tasks and workers increases. In other words, the problem size and complexity increase. Therefore, to reduce the problem size and complexity, we employ a decomposition technique, which enables separate optimization of task–worker assignments and task completion orders of workers through iterative interactions. Moreover, to address the problem, a deep Q-learning (DQN)-based algorithm, which is called assignment DQN (A-DQN) along with a local search (A-DQN w/LS), is proposed.

Deep Q-learning is a reinforcement learning (RL) [18] algorithm, which is an interactive learning technique and has shown its success in solving complex optimization problems [19]. In RL, the agent tries out different actions in each state, observes the reward, and decides on the action that maximizes the cumulative reward. Moreover, in each step, RL adds the expected reward from future states to the reward of the current state, effectively influencing the current decision towards a potential solution. As a result, under a certain assignment, RL can decide which worker selection will lead to a better solution. Therefore, A-DQN, along with local search, can efficiently solve the considered problem through iterative and local improvements, which is verified by conducting numerical simulations. Specifically, the proposed method is compared with other MCS approaches under various situations, using the total cost (the sum of the traveling cost and sensing cost of all tasks) and the CPU time as the measure of effectiveness, where A-DQN w/ LS obtains better performance with a reasonable running time.

In our previous paper [20], although a similar problem has been considered, it has limitations. First, a metaheuristic approach, i.e., a memetic genetic algorithm for task allocation (MGATA), is used to find the task–worker assignment, which is more prone to fall into local optima since population-based algorithms are easy to lose genetic diversity due to selection pressure and genetic drift [21]. Second, one job is divided into multiple tasks that have the same deadline, which is inefficient in a large-scale platform where heterogeneous tasks are published simultaneously. Third, the energy consumption model of UAVs is not considered, whereas, in this paper, we consider energy constraints of UAVs, i.e., a UAV should be assigned to tasks if it can complete the tasks and return with the available energy.

The main contributions of this paper are summarized as follows:

- This paper decomposes the considered problem into an assignment problem and a set of task completion order problems to reduce the problem size and enable diverse searching strategies. The assignment problem and the task completion order problem are formulated as integer linear programming (ILP) problems with the aim of minimizing the total cost of performing tasks subject to the constraints; deadline, required sensor, required workers' type, and the energy constraints of the UAV.
- To solve the task assignment and routing problem, a deep Q-learning-based algorithm called assignment DQN with local search is proposed, where an asymmetric traveling salesman (ATSP) heuristic is iteratively employed to find the workers moving paths. Assignment DQN (A-DQN) guides the algorithm towards the best solution by maximizing the aggregated rewards, whereas the local search handles the long training time and locally searches the solution space for a better option.
- To evaluate the proposed algorithm, simulations are conducted using different experimental scenarios, where A-DQN w/ LS shows its efficiency by achieving a lower total cost than existing methods. In addition, we compare different DQN architectures and show the effects of adding a local search with the DQN.

The rest part of this paper is organized as follows. Section II discusses related works. The system model and problem formulation are presented in Sections III and IV, respectively. Section V describes the details of our proposed algorithm. We show the results of the simulation in Section VI, and Section VII concludes the paper.

II. RELATED WORKS

Recently, there have been a lot of studies on MCS task allocation, taking various task allocation scenarios into account. For example, some studies have focused on single-task allocation [9]–[11]. In these studies, one task is distributed at some point of interest within the given sensing area, and the sensing period is usually divided into equal-length cycles. The sensing data are collected at each cycle by a subset of workers with the aim of maximizing the spatial–temporal coverage. On the contrary, we consider an MCS system where heterogeneous time-constraint tasks are published concurrently, and multiple tasks can be assigned to a worker who needs to intentionally visit the task venues to perform the tasks. Therefore, these existing works are very different and cannot be applied to the model considered in this work.

Another group of works has proposed task allocation strategies considering multi-task allocation scenarios and time-sensitive tasks. For example, the study in [12] proposed a task allocation framework with the goal of efficiently selecting a set of workers such that high-quality results are obtained for each task within the given time. In [13], the authors aimed to maximize the quality of interest (QoI) of each task while minimizing its completion time. They took distance, workers' reputation, and confidence level into account to calculate QoI, where the confidence level is calculated by using the workers' historical mobility. Guo *et al.* [14] devised a multi-task assignment framework for two different sensing environments: workers selection for time-sensitive tasks, which requires workers to visit the task venue intentionally, and delay-tolerant tasks, where workers are not required to move purposefully. However, these studies do not consider the task completion order or moving path of the workers, which play an important role in satisfying the time-constraint of the tasks.

A few studies have considered the worker's task completion order. For instance, in [15], Gong *et al.* focused on maximizing the sum of tasks' quality while taking the movement path of the worker into consideration. Zhao *et al.* [16] wanted to find a task allocation approach such that the total number of completed tasks is maximized. They also consider tasks and workers' deadlines (i.e., workers should reach their destination before the deadline). The authors in [17] investigated the

effect of time constraints on multi-task allocation, and aimed to maximize the utility of the platform while considering the worker's working deadline and the task's deadline. Above mentioned studies consider the worker's path or task completion order either as a simple path along with the workers' daily route or as scheduling without any cycle.

In contrast, in our work, the worker's path is a Hamiltonian cycle, and the lowest-cost path is obtained by solving a TSP problem, which is more practical since workers perform tasks in the real-world. Furthermore, we decompose the original problem into several relatively simpler problems to reduce the problem size and enable separate optimizations of different problems, and employ an RL-based solution framework to find the task–worker assignments along with the workers moving paths.

III. SYSTEM MODEL

We assume that multiple jobs can be published at the same time, and each job consists of multiple sensing tasks based on the location. Let $\mathbb{T} = \{1, \ldots, |\mathbb{T}|\}$ and $\mathbb{W} = \{1, \ldots, |\mathbb{W}|\}$ be the sets of tasks and workers, respectively. $\mathbb{W}$ contains two types of workers: UAV (set of UAVs is denoted by $\mathbb{W}^1$) and human (set of human workers is denoted by $\mathbb{W}^2$) where $|\mathbb{W}| = |\mathbb{W}^1| + |\mathbb{W}^2|$.

Each task $j(j \in \mathbb{T})$ is associated with a location L_j, required sensors set $\mathbb{R}_j$, the time needed to perform the task ζ_j, payment for performing the task with respect to time η_j, sensing frequency f_j, size of the sensed data b_j, required worker type w_j, and deadline d_j, by which the assigned worker has to reach the task venue and complete the task. We consider three types of requirement for the worker: 1) some tasks may require only UAV (i.e., $w_j = 1$), 2) some tasks may require only human (i.e., $w_j = 2$) and 3) some tasks may not have any specific requirement, i.e., both UAV and human can perform the task ($w_j = 0$).

Each worker $i(i \in \mathbb{W})$ has a description comprising the current location l_i, available sensors set $\mathbb{A}_i$, velocity V_i, base payment with respect to distance p_i, and worker type $v_i(v_i \in \{1, 2\})$. If the worker is a UAV (i.e., $v_i = 1$), the description contains additional specifications: energy consumption for taking-off and landing E^i_{tl} (which can vary depending on the altitude and velocity of the worker), sensing and transmitting e^i_{st}, horizontal flying e^i_f, and hovering e^i_h. Furthermore, the worker description of a UAV also contains battery capacity e^i_R.

IV. PROBLEM FORMULATION

The considered task–worker assignment problem contains a set of workers' path-planning problems, which makes it very complex to solve. Therefore, we decompose the original problem into an assignment problem (i.e., the main problem) and a set of task completion order problems (i.e., subproblems) by employing a decomposition technique similar to one from [20]. In this section, the main problem and the subproblems are formulated as two ILP problems where the main objective is achieved through iterative interactions between them. Specifically, the main problem assigns workers to tasks and passes assignments to the subproblems. Each subproblem minimizes its own total cost and returns the sum of all the workers' total costs to the main problem, which then assesses the current assignment.

To incorporate the current positions of the workers (since workers will visit task venues following the Hamiltonian cycle), we first introduce a dummy task set $\mathbb{T}_d$, which is generated based on the registered workers' information. Specifically, the number of elements in the dummy task set is the same as the worker set, i.e., $\mathbb{T}_d = \{1, \ldots., |\mathbb{W}|\}$. Location, required worker type, and required sensors of each dummy task j are the same as the current location, worker type, and available sensors of worker i, respectively, i.e., $L_j = l_i$, $w_j = v_i$, and $\mathbb{R}_j = \mathbb{A}_i$ where $j = i$. In addition, for each dummy task $j(j \in \mathbb{T}_d)$, task duration ζ_j, sensing frequency f_j, and data size b_j are set to 0, whereas deadline d_j is set to a large positive number. Now, let us define a task set $\mathbb{T}'$ that contains the sets of dummy tasks $\mathbb{T}_d$ and actual tasks $\mathbb{T}$, where $\mathbb{T}' = \{1, \ldots.|\mathbb{T}'|\}$ and $|\mathbb{T}'| = |\mathbb{T}_d| + |\mathbb{T}|$.

Each UAV is battery-operated and consumes energy for traveling through task venues and performing tasks. Let E^{ij}_f, E^{ij}_h, and E^{ij}_c denote the energy consumption for flying horizontally, hovering, and communication, respectively. For each task j, E^{ij}_f, E^{ij}_h and E^{ij}_c of UAV $i(i \in \mathbb{W}^2)$ are calculated in (1)-(3):

$$E^{ij}_f = c_{ij} \times e^i_f \tag{1}$$

$$E^{ij}_h = \zeta_j \times e^i_h \tag{2}$$

$$E^{ij}_c = e^i_{st}(b_j \times f_i \times \zeta_j) \tag{3}$$

where c_{ij} is the traveling distance between the location of worker i and task j. Definitions for other notations used in problem formulation are presented in Table 1.

Decision variables used in the problem formulation are defined as follows:

$$y_{ij} = \begin{cases} 1, & \text{if worker } i \text{ is assigned to task } j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$x_{ijk} = \begin{cases} 1, & \text{if worker } i \text{ visits task } k \text{ right after} \\ & \text{visiting task } j. \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

The main problem is minimizing the total cost of all workers, where M_i denotes the total cost of worker i. Then, the main problem can be expressed as:

$$\min \sum_{i \in \mathbb{W}} M_i \tag{6}$$

$$\text{subject to}: \sum_{j \in \mathbb{T}'} y_{ij} \geq 0, \quad \forall i \in \mathbb{W} \tag{7}$$

$$\sum_{i \in \mathbb{W}} y_{ij} = 1, \quad \forall j \in \mathbb{T},\ \mathbb{T} \subseteq \mathbb{T}' \tag{8}$$

$$y_{ij} = 1, \quad \forall j \in \mathbb{T}_d,\ \forall i \in \mathbb{W},\ j = i, \mathbb{T}_d \subseteq \mathbb{T}' \tag{9}$$

TABLE 1. Model parameters.

Task parameters	
$\mathbb{T}$	Set of all tasks
d_j	Deadline of task j
ζ_j	Task duration of task j
$\mathbb{R}_j$	Set of required sensors for task j
L_j	Location of task j
w_j	Worker type required to perform task j
f_j	Sensing frequency of task j
b_j	Sensed data size of task j
η_j	Salary for the task j
Worker parameters	
$\mathbb{W}$	Set of all workers
$\mathbb{W}^1$	Set of UAV workers
$\mathbb{W}^2$	Set of human workers
v_i	Type of worker i
$\mathbb{A}_i$	Set of available sensors in a worker's device
l_i	Location of worker i
V_i	Velocity of worker i
p_i	Worker's base payment w.r.t distance
e^i_R	Battery capacity of UAV i
Other	
$\mathbb{T}_i$	Set of tasks assigned to worker i, $\mathbb{T}_i = \{0, ...\lvert\mathbb{T}_i\rvert\}$ and $\mathbb{T}_i \subseteq \mathbb{T}'$, where 0 denotes the dummy task assigned to the worker.
c_{jk}	Distance between tasks j and k.
z	A large number

$$|\mathbb{R}_j \cap \mathbb{A}_i| + (1 - y_{ij}) \geq 1, \quad \forall j \in \mathbb{T}' \tag{10}$$

$$(w_j - v_i)y_{ij} = 0, \quad \forall j \in \mathbb{T}', \ \forall i \in \mathbb{W}, \ w_j \in \{1, 2\} \tag{11}$$

The total cost M_i of worker i is defined as the sum of the traveling cost and the sensing costs of the tasks assigned to the worker. Therefore, M_i of each worker i is obtained by solving the corresponding subproblem, i.e., the task completion order problem. The subproblem is converted to an asymmetric traveling salesman problem (ATSP) by adding sensing costs with the traveling costs. Specifically, we calculate the cost of visiting task k from j as the sum of the sensing cost and traveling cost. The sensing cost can be different for each task, which makes the cost of the path between tasks j and k different depending on the direction. For example, assume that a worker performs two tasks: t_1 and t_2. Sensing duration (ζ_j) and sensing payment per unit time (η_j) of task t_1 are 10 and 4, respectively, whereas for t_2 they are 20 and 3, respectively. The distance between t_1 and t_2 is 1 unit, and the worker's base payment per unit distance (p_i) is 30. Then, the cost of traveling to t_2 from t_1 is 70 ($30 \times 1 + 10 \times 4$) and from t_2 to t_1 is 90, which is asymmetric. Therefore, the i^{th} subproblem, i.e., worker i's task completion order problem, is formulated as:

$$\min M_i = \sum_{j \in \mathbb{T}_i} \sum_{k \in \mathbb{T}_i} \Big((p_i \times c_{jk}) + (\zeta_k \times \eta_k) \Big) x_{ijk} \tag{12}$$

$$\text{subject to} : \sum_{k \in \mathbb{T}_i} x_{ijk} = 1, \qquad \forall j \in \mathbb{T}_i \tag{13}$$

$$\sum_{j \in \mathbb{T}_i} x_{ijk} = 1, \qquad \forall k \in \mathbb{T}_i \tag{14}$$

$$u^i_0 = 1 \qquad j \in \mathbb{T}_i \tag{15}$$

$$2 \leq u^i_j \leq |\mathbb{T}_i|, \quad \forall j \in \mathbb{T}_i \tag{16}$$

$$u^i_j - u^i_k + 1 \leq (|\mathbb{T}_i| - 1)(1 - x_{ijk}), \quad \forall j, k \in \mathbb{T}_i \setminus 0 \tag{17}$$

$$\sum_{j'=1}^{(u^i_{j'=k})-1} \left[\frac{c_{j^*k^*}}{V_i} + \zeta_{j^*} \Big|_{j^* | u^i_{j^*} = j', k^* | u^i_{k^*} = j'+1} \right] \leq d_k - \zeta_k, \quad \forall k, j^*, \quad k^* \in \mathbb{T}_i \setminus 0 \tag{18}$$

$$E^i_{tl} + \sum_{j \in \mathbb{T}'} \sum_{k \in \mathbb{T}', j \neq k} \Big(E^{jk}_f + E^{ij}_h + E^{ij}_c \Big) x_{ijk} \leq e^i_R, \quad i \in \mathbb{W} \setminus \mathbb{W}^2 \tag{19}$$

where u^i_j is the number of tasks performed by worker i on the way to task j.

Equation (6) represents the objective function of the formulation. Constraints (7) and (8), respectively, ensure that workers perform a non-negative number of tasks, and each task is assigned to exactly one worker. Equations (8) and (9) together ensure that each worker is assigned to exactly one dummy task that has the same location as that worker. Inequality (10) ensures that the task is assigned to the worker who has all the required sensors. Constraint (11) ensures that the task is assigned to an appropriate worker who matches the task's required worker type. Equation (12) is the objective function of the task completion order problem of worker i. Constraints (13) and (14) make sure that each task is visited exactly once. Subtour formation in the trajectories of the worker is prohibited by constraints (15), (16), and (17) together following the Miller-Tucker-Zemlin (MTZ) subtour elimination method [22]. The set of subtour elimination constraints and inequality (18) combined make sure the worker assigned to each task reaches the task venue before the task start time. Constraint (19) ensures that the total energy consumption of a UAV does not exceed its battery capacity.

The original problem comprises two different optimization problems, which contains two different decision variables, i.e., y_{ij} and x_{ijk}. Thus, the total number of variables is $|\mathbb{W}| \times |\mathbb{T}'| + |\mathbb{W}| \times |\mathbb{T}'| \times |\mathbb{T}'|$, and the total number of 0-1 combinations in the problem is $2^{|\mathbb{W}| \times |\mathbb{T}'| + |\mathbb{W}| \times |\mathbb{T}'| \times |\mathbb{T}'|}$; for large values of tasks $|\mathbb{T}|$ and workers $|\mathbb{W}|$, the space complexity of the original problem becomes extremely high. Therefore, we reduce the problem size by decomposing the problem into the main problem and a set of subproblems, where the main problem and the subproblems are solved separately. After decomposition, the assignment problem (i.e., the main problem) and each subproblem (the ATSP problem) contain $|\mathbb{W}| \times |\mathbb{T}'|$ and $|\mathbb{T}_i| \times |\mathbb{T}_i|$ variables, respectively, whereas the maximum value of $|\mathbb{T}_i|$ can be $|\mathbb{T}| + 1$.

However, the decomposed problem still contains a large solution space. Assume that a worker can perform a total of $|\mathbb{T}|$ tasks. Then, there will be $2^{|\mathbb{T}|+1}$ ways to assign a worker, and $(|\mathbb{T}|+1)!$ different possible orders for completing the tasks for each worker (in the worst-case scenario). Thus, for all workers $|\mathbb{W}|$, the total number of possible solutions will be $(2^{|\mathbb{T}'||\mathbb{W}|}(|\mathbb{T}|+1)!$, which increases exponentially with the increase in the numbers of tasks and workers. Furthermore, each subproblem and the main problem are formulated as an ATSP problem and an assignment problem, respectively, where both of them are well-known NP-hard problems. Therefore, the above-formulated problem is also an NP-hard problem.

V. DRL-BASED TASK ALLOCATION FRAMEWORK

In this section, a deep reinforcement learning (DRL)-based task assignment algorithm is proposed to assign $|\mathbb{T}|$ sensing tasks to $|\mathbb{W}|$ workers (i.e., people or UAVs) such that the total cost is minimized and the given constraints are satisfied. Fig. 2 illustrates the framework of the proposed strategy. First, the deep Q-learning algorithm or deep Q-network (DQN) takes the task and worker sets as input, trains the agent for a given number of episodes while maximizing the cumulative reward, and outputs the best solution, i.e., the task–worker assignment. Then, the local search takes the best solution obtained by the DQN as input and finds the final task–worker assignment, which is the final output. In other words, the DQN guides the algorithm towards a better solution, and the local search locally optimizes it. Adding a local optimizer at the end of the DQN saves training time since it needs a lot of training steps to find the best solution [18], [23].

For the rest of this section, we first present the environment design, including state, action, and reward. Then, the DQN-based task allocation scheme with Q-network training is described in detail.

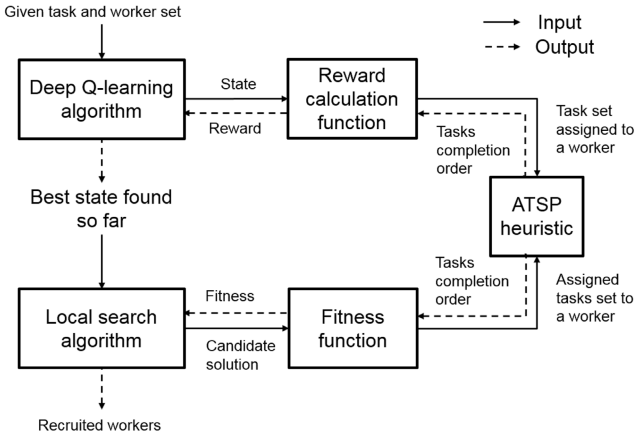

FIGURE 2. Framework of the proposed approach.

A. MDP MODEL

Generally, RL problems can be regarded as a Markov Decision Process (MDP) in which the next state is determined by the current state and the action taken on it [24]. Therefore, we formulate the task–worker selection problem as an MDP defined by the tuple $< \mathbb{S}, \mathcal{A}, \mathcal{R} >$ following the interaction model between the MCS server and the environment. At each step t, the agent observes state $s \in \mathbb{S}$, takes action $a \in \mathbb{A}$, moves to the next state s', and receives reward $r \in \mathcal{R}$.

1) STATE AND ACTION SPACES

In this work, the MCS server is the agent, and task–worker assignment matrix $S \in \mathbb{B}^{|\mathbb{T}|\times|\mathbb{W}|}$ represents the state of the environment. If worker i is selected to perform task j, $S_{i,j} = 1$; otherwise, $S_{i,j} = 0$. The task–worker assignment matrix is converted to a one-dimensional vector, which is denoted by s, and the size of the state space is $2^{|s|}$.

The action is assigning a worker to a task. Note that a task needs to be first determined for the worker assignment. Thus, the action consists of two phases: 1) task-selection and 2) worker assignment. In the task-selection phase, subaction j (i.e., task j) is selected. In the worker assignment phase, subaction i is selecting a worker (i.e., worker i is chosen for task j). If worker i is selected for task j, the rest of the workers cannot be assigned to task j. We use $A = \{j, i\}$ to denote the action, and the size of the action space is $|\mathbb{A}|(\mathbb{A} = \{1, \ldots, |\mathbb{W}| \times |\mathbb{T}|\})$.

2) IMMEDIATE REWARD

The immediate reward, $r = R(s, a, s')$, which is received by executing an action a at state s, indicates how good the action is. The objective of the task allocation problem (minimization of the total cost while ensuring the given constraints) should be reflected in the reward function. In addition, for faster learning, a penalty can be added to the reward to prevent the agent from going to infeasible states and to states with high total costs. Therefore, reward r is defined as:

$$r = \begin{cases} m\left(\dfrac{z - T_c(s')}{z}\right), & \text{if all constraints are feasible} \\ m\left(\dfrac{z - T_c(s')}{z} \times \alpha\right), & \text{if all constraints are feasible and } T_c \text{ is higher in } s' \text{ than } s \\ m\left(\dfrac{z - T_c(s')}{z}\right) \times (1 - (\beta\Phi)), & \text{if constraints (8), (18), or (19) is violated} \end{cases} \tag{20}$$

where $T_c(s)$ is the total cost in a state s, z is a large positive number, $\alpha(0 \leq \alpha \leq 1)$ is a coefficient, $m(m > 1)$ adds flexibility to the magnitudes of the reward, and Φ is the sum of the following: the ratio of tasks that violate the deadline constraint, the ratio of UAVs that violate the energy constraint, and the ratio of tasks that do not have any worker assigned. β is a weight and $\beta > 0$. When the worker resource is rich and the environment is less constrained (e.g., a loosen

deadline), a greater value of β, which leads to a large penalty, encourages the agent to take actions that lead to states with higher rewards, making learning faster. However, in the opposite scenario, too much of a penalty or a large value of the β results in a lot of negative rewards in the environment, which increases the training time. In addition, since the DQN agent may suffer from the long training problem in the environment with a varied scale of rewards [18], Φ is clipped to the range $(0, m)$. As a result, the scale of error derivatives can be limited to a certain range, which can allow more stable training of the Q-network.

B. THE DQN-BASED TASK ALLOCATION ALGORITHM

The goal of the RL algorithm is to learn an optimal policy π through trial and error, which maximizes the Q-value function $Q(s, a)$ at state s:

$$Q(s, a) = \mathbb{E}[R(s, a) + \gamma Q(s', a')|\pi] \tag{21}$$

where γ is the discount factor that determines the agent's interest in the future environment's state. In traditional RL, tabular Q-learning is widely used to estimate the Q-value function [25], [26]. However, it stores the Q-value of each state–action pair in a Q-table or matrix, the size of which increases dramatically when the state and action space increases, leading to high space and time complexity. Thus, to overcome this problem, the DQN is proposed by DeepMind [18], which exploits a neural network (NN) to estimate the Q-value of each state–action pair instead of keeping the Q-table.

In this work, we use DQN to learn the Q-value of an action A at state s, $Q(s, A)$. In particular, DQN takes the state as input of the NN and outputs Q-values for all state–action pairs $Q(s, A)$ (a linear output layer without an activation function), as shown in Fig. 3(a). In DQN, since the number of output neurons in the NN is equal to the action space, the number of output nodes can be huge depending on the number of tasks and workers. Therefore, in some utmost cases, there could be no weight update for certain output nodes for a long time because the corresponding action has not been taken before. Then, the Q-value estimation of the node will be totally random. As a result, DQN may not generalize well over previously unseen states, i.e., it is likely to converge to a local optima or may require a long training time to find the best state.

An alternative approach is estimating the Q-value of workers for selected task j. Specifically, one subaction (i.e., a task) with the state is used as an input of the NN, and the output is the Q-value of each worker, as shown in Fig. 3(b). We call this architecture the task DQN (T-DQN) for convenience. Although this architecture reduces the number of neurons in the output layer, there could still be no example of a certain action when there is a large number of workers; hence, no weight update.

Another way to handle this problem is to consider a DQN architecture, where the number of output nodes is independent of the possible number of actions, i.e., the NN

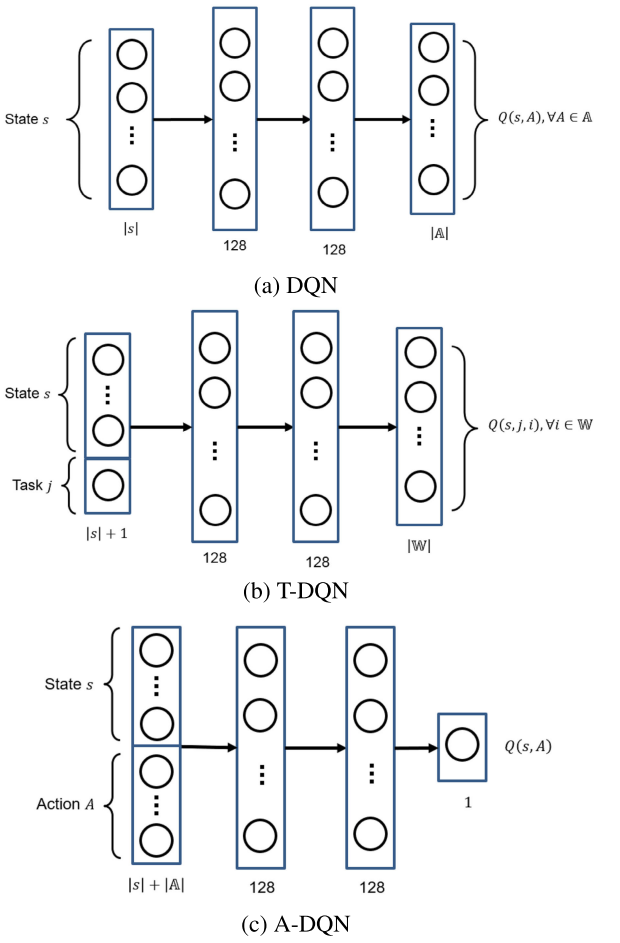

FIGURE 3. Different deep Q-learning architectures.

outputs the single Q-value approximation for each possible action [27]. In our work, this architecture is called assignment DQN (A-DQN). The input of the NN in A-DQN is a state and task–worker assignment (i.e., an action represented as a one-hot vector), and the output is the Q-value approximation of the corresponding action, as shown in Fig. 3(c). In A-DQN, there is only one neuron in the output layer, which alleviates the problems of random Q-value estimations for certain actions. As a result, it generalizes better on the action that has never been chosen before and needs less experience to find a good solution as the output. Furthermore, the combination of the state and one-hot vector of the action as the input of NN helps the model learn the relations between the current state and the selected task and worker more precisely.

However, using an NN to train the agent to learn the Q-value approximation is computationally expensive, i.e., the agent needs to obtain a great number of samples to find the best solution. Therefore, to reduce the training time, a local search algorithm is applied at the end of the A-DQN algorithm to locally optimize the best solution obtained after training the A-DQN for a given number of episodes. The solution representation of the local search and the fitness of

the solution, respectively, are the same as the state and the reward of the DQN.

Since the performance of local searches (e.g., tabu search or simulated annealing (SA)) can vary depending on the problem types and sizes [28], [29], two variants of the A-DQN-based task allocation and worker routing algorithm, called A-DQN with local search (A-DQN w/ LS), i.e., A-DQN w/ tabu and A-DQN w/ SA, are proposed.

- A-DQN w/ tabu: In this algorithm, after the DQN is trained, a tabu search [30] is employed to locally optimize the solution obtained by the A-DQN, which is the initial solution of the tabu search. At each iteration, a new solution from the set of the neighboring solutions of the current best solution is selected. If two solutions differ by one element, they are considered neighbors. As the neighborhood directly affects the results of the local search, the following strategies are used to create the neighborhood. We create half of the neighborhood by randomly selecting one task and a worker for that task. The rest of the neighborhood is created randomly by flipping the value of one element. The new solution is accepted if its fitness is better than the fitness of the current best solution and not in the tabu list, which contains the task–worker assignment matrix and the element number.
- A-DQN w/ SA: This algorithm uses simulated annealing [31] as a local search algorithm for local optimization of the solution obtained from A-DQN. In each iteration, one new solution, which is the neighbor of the current solution, is created by following the same strategies as the tabu search. If the fitness of the new solution is superior to the current solution, then it is accepted; otherwise, the new solution is chosen with a certain probability. This probability of acceptance decreases as the number of iterations increases.

The pseudocode of the A-DQN w/ tabu or w/ SA is presented in Algorithm 1. At first, parameters of the Q-network and the target-network are initialized in Line 3. Note that DQN uses two different networks with different parameters to estimate the target Q-value and the Q-value of the current state, whereas both networks use the same NN architecture. The NN consists of two fully connected hidden layers with a tanh activation function since fully connected layers can handle heterogeneous input and catch the correlations between state and action. The number of units in the first and second hidden layers is set to 128. The parameters of Q-network and target-network at iteration t are denoted by θ_t and θ_t^-, respectively.

Similar to acting in games, during each episode, the agent starts from the initial state and plays L_{iter} steps, where L_{iter} is the length of an episode (lines 4, 5, and 6). At the beginning of each step, the current state is fed to the Q-network, and the Q-values of all actions are obtained. Then, the agent chooses an action A using an ϵ-greedy policy to balance the exploration and exploitation, where the ϵ value gradually

Algorithm 1 A-DQN-Based Task Allocation and Worker Routing Algorithm

Input: Set of tasks $\mathbb{T}$ and set of workers $\mathbb{W}$

1: $n_{steps} = 0$ ▷ Total number of steps that agent has played
2: Initialize samples (s, A, s', r) in a pool
3: Initialize parameters θ_t and θ_t^- of the Q-network and target network
4: **for** $l \leftarrow 1$ to N_{iter} **do** ▷ N_{iter}: Number of episodes
5: current state $\leftarrow$ initial state s_0
6: **for** $t \leftarrow 1$ to L_{iter} **do** ▷ L_{iter}: one episode length
7: $s \leftarrow$ current state
8: choose an action A using ϵ-greedy policy
9: perform action A in state s and observe next state.
10: $s' \leftarrow$ next state
11: calculate reward r using (20)
12: add new sample (s, A, s', r) to the pool
13: randomly select η samples from the pool
14: calculate the Q-value of the next state using target network.
15: update θ_t using selected η samples ▷ Eq. (22)
16: current state $\leftarrow s'$
17: $n_{steps}++$
18: **if** n_{steps} mod $c == 0$ **then**
19: $\theta_t^- \leftarrow \theta_t$ ▷ Copy parameters from Q-network to target-network
20: **end if**
21: **end for**
22: test periodically and record the best state
23: **end for**
24: apply tabu search or SA to the best state found so far

decreases over the training steps. Next, the agent performs the action, obtains the next state s' and reward r (lines 7-11).

To reduce correlations between data samples, a scheme called experience replay is implemented by storing the agent's experience, $e_t = (s_t, A_t, r_t, s'_t)$, in a memory pool at Line 12. At each step in the iteration t, η samples (s, A, s', r) are randomly taken from the pool to update the Q-network parameters θ_t (lines 13-15) by minimizing the following mean squared loss function:

$$L_t(\theta_t) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', A'; \theta_t^-) - Q(s, A; \theta_t)\right)^2\right] \tag{22}$$

The loss function represents the difference between the target Q-value estimated using the target network, i.e., $r + \gamma \max Q(s', A'; \theta_t^-)$, and the Q-value obtained from the Q-network, $Q(s, A; \theta_t)$. The Adam optimizer with the learning rate $\alpha = 0.001$ is used to update the model parameters of the Q-network.

The target-network parameters θ_t^- are copied from the Q-network parameters θ_t every c steps (line 19) and remain unchanged between consecutive copies. In addition, we test the agent periodically and record the best state.

Lastly, a local search (i.e., tabu or SA) is applied to the solution obtained by deep Q-learning. In particular, the best solution recorded by periodically testing the agent is used as the input of the local search (line 24).

Immediate reward r is calculated by solving the subproblems, i.e., the task completion order of the workers. First, an ATSP heuristic, iterated 3opt (i3opt) [32], is applied to obtain the task completion order of each worker. Next, the values of Φ and $T_c(s)$ are obtained by using the task completion orders of the workers, and then reward r is calculated using Equation (20).

VI. PERFORMANCE EVALUATION

In this section, the proposed algorithms are evaluated by conducting simulations. We first introduce the simulation setup and algorithms for comparison. Then, the performances of the proposed methods are compared with other approaches with respect to total cost (i.e., the objective function value) and CPU time under various dynamics, such as different numbers of iterations, tasks, workers, and deadlines.

A. SIMULATION SETUP

Table 2 summarizes the parameters for the experiments. It is assumed that each worker is equipped with a specific set of sensors, and a task may require up to three types of sensors. In each simulation, the arrival deadline, sensing duration of the tasks, and the base unit payment of the workers are set randomly within the range. The default values are denoted by bold fonts in Table 2. Recall that the deadline of a task is the time for the worker to reach the location and perform the task. Therefore, we use two different parameters: the arrival deadline, which is the required time within which a worker should reach the task location, and the task duration. For simplicity, sensing frequency and data size are set to the same for all tasks. All UAVs are also assumed to have the same specifications.

TABLE 2. Experimental settings.

Parameter	Value
Number of workers (w)	{ 6, **10**, 14, 18, 22}
Number of tasks (k)	{6, **8**, 10, 12}
Area size	7x7 km^2
Arrival deadlines of the jobs (in minutes)	[5, 10], [10, 20],**[20, 30]**, [30, 40], [40, 50]
Task durations (in minutes)	**[20, 30]**
Sensed data size	10kb
Sensing frequency (per minute)	0.5
UAV's velocity (km/h)	**55**
Human's velocity (km/h)	**50**
Battery capacity of a UAV	50×10^4 joules
Energy consumption for sensing and transmitting one bit	0.4 nJ/bit
Energy consumption for flying horizontally	100 J/s
Energy consumption for taking-off and landing	100 J/s
Energy consumption in hover mode	100 J/s
Base payments of workers	**[20, 40]**

A taxi movement dataset called roma/taxi [33] is used to determine the locations of tasks and workers. The dataset provides the GPS points of taxi drivers in the city of Rome during one month period from February to March in 2014. Since the GPS traces are collected from the different parts of the city, we first select a 7 km x 7 km crowded region in the city and use this region as the simulation area. Then, the locations of tasks and workers are randomly selected from the list of taxi drivers' GPS points inside the selected region.

Experiments are performed on a platform equipped with an Intel Xeon E5-1620 v3 CPU @ 3.50GHz, with an NVIDIA GeForce GTX TITAN X graphics card and 16 GB of RAM. All DQN-based algorithms are implemented in Tensorflow's GPU version.

In this section, first, we compare the performance of three different DQN architectures (i.e., DQN, T-DQN, and A-DQN) in terms of the total cost. Then, the baseline A-DQN is compared with the variants of A-DQN w/ LS (i.e., A-DQN w/ tabu and A-DQN w/ SA) in terms of total cost and CPU time. In A-DQN w/ tabu and A-DQN w/ SA, after A-DQN is trained, the local search algorithm (tabu or SA) is run ten times and the best value is obtained. For fairness, parameters of tabu and SA, such as the number of iterations and the neighborhood size, are selected in such a way that the total searches in solution space for both algorithms are approximately the same. For example, we stop the tabu search if there is no improvement for five iterations with a neighborhood size of 100, and stop SA if there is no improvement for 500 iterations.

Then, the performances of A-DQN w/ tabu and A-DQN w/ SA are compared by using the evaluation metric total cost for different parameters, such as the number of tasks and workers, and the arrival deadline. We also compared the proposed methods with other MCS approaches: MATC-IGA, and MGATA. Those algorithms are stopped if there is no improvement for a given number of iterations. In addition, MATC-IGA and MGATA, described as follows, are both run ten times and the best value is obtained.

- MATC-IGA [17]: A genetic algorithm is employed to find the recruited workers set where the initial population is generated by using a random-greedy algorithm. MATC-IGA uses a repair operation to obtain a valid chromosome from the produced invalid chromosome through crossover and mutation.
- MGATA [20]: In this study, a memetic genetic algorithm is applied to find the task–worker assignments and uses an ATSP heuristic to obtain the worker's task completion order. However, because of the different assumptions and the considered scenario, MGATA cannot be directly applied to our problem. Therefore, we adjusted the algorithm according to our problem formulation.

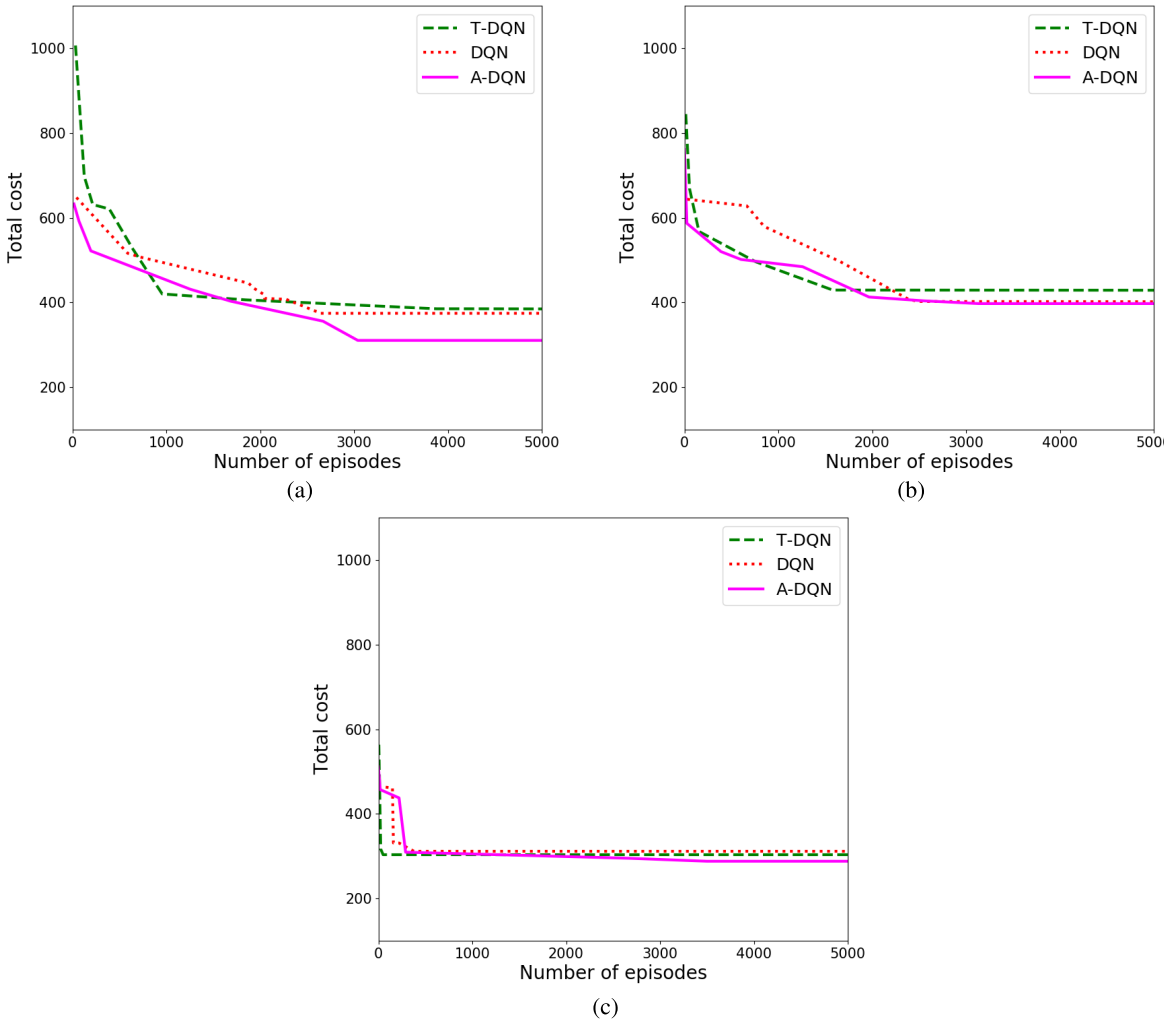

FIGURE 4. Total cost learning curves of different DQN architectures along with the increases in the number of episodes for: (a) eight tasks and 10 workers; (b) 10 tasks and 10 workers; (c) eight tasks and 18 workers.

B. RESULTS EVALUATION AND ANALYSIS

Here, the numerical results from the simulations are presented to evaluate the performances of the proposed approaches, where the result of the greedy algorithm [20] is used as the initial state of each approach. In addition, each simulation is averaged over three runs.

1) PERFORMANCE COMPARISON BETWEEN DIFFERENT DQN ARCHITECTURES

Fig. 4 shows the convergence process of the three DQN architectures (DQN, T-DQN, and A-DQN) along with the increase in the number of episodes under different numbers of tasks and workers. From the figure, we can see that the total cost obtained by all three architectures decreases as the number of episodes increases. This is because the agent initially explores the environment more and then gradually learns to make decisions, which maximizes the total aggregated reward, i.e., leads to a state with a lower total cost. Among them, A-DQN obtains the lowest total cost, which can be seen in Figs. 4(a), (b), and (c). The reason is that the A-DQN has only one node in the output layer resulting in more efficient generalization, and the ability of generalization enables the algorithm to find a relatively better solution within a smaller number of iterations. It can be seen that for all cases, DQN and T-DQN do not generalize well, i.e., do not reach a state with a total cost lower than A-DQN, which shows the effect of the over-sized output layer.

2) PERFORMANCE COMPARISON BETWEEN BASELINE A-DQN AND A-DQN W/LS

The effect of adding a local search to the DQN with respect to the total cost and CPU time is shown in Fig. 5. A-DQN with two different local search algorithms, i.e., tabu search and SA, is compared with A-DQN, which was chosen as the baseline because it outperforms DQN and T-DQN.

First, Figs. 5(a)-(c) show the total cost obtained by each algorithm for different numbers of episodes of A-DQN

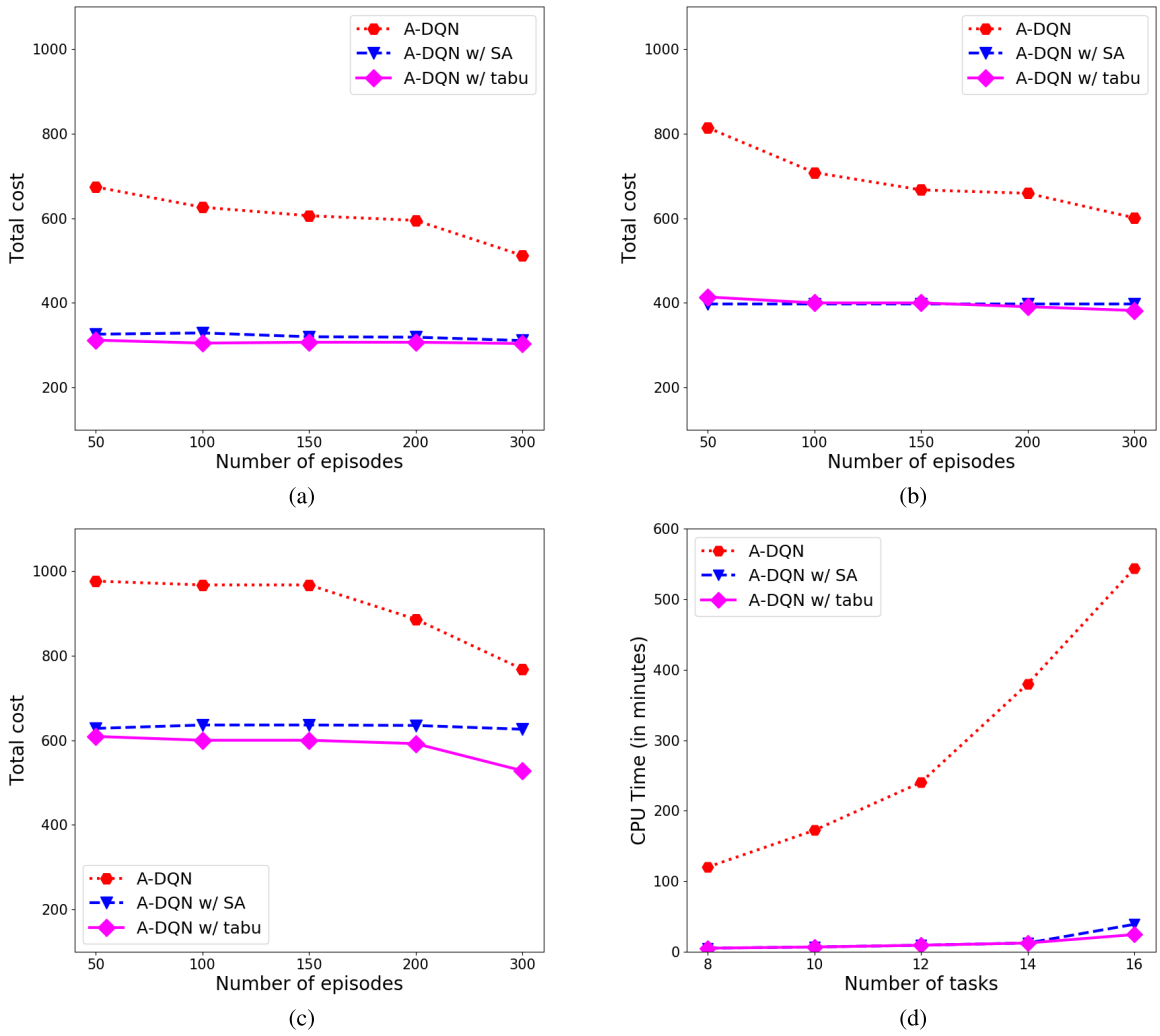

FIGURE 5. Performance comparison of the baseline A-DQN and A-DQN with local search: A-DQN w/ tabu and A-DQN w/ SA. (a)-(c) Total cost obtained by the methods for different numbers of iterations of A-DQN under different numbers of tasks and workers: (a) eight tasks and 10 workers; (b) 10 tasks and 10 workers; (c) 14 tasks and 10 workers. (d) Time complexity of the approaches given total costs for different numbers of tasks.

(i.e., 50, 100, 150, 200, and 300). Note that in A-DQN w/ tabu and A-DQN w/ SA, only the episodes of A-DQN changes while iterations in tabu and SA remain unchanged. As can be seen in the figures that A-DQN with a local search (i.e., A-DQN w/ tabu or A-DQN w/ SA) obtains a lower total cost than the baseline A-DQN. Among the two local searches, A-DQN w/ SA does not show significant improvement when the quality of the input increases, whereas A-DQN w/ tabu shows a slightly greater change in the obtained total cost and outperforms A-DQN w/ SA as the training episodes increases.

Then, the time each method needs to obtain the given total costs for the different number of tasks is shown in Fig. 5(d). Each baseline cost is obtained by training the baseline A-DQN for 1500 episodes. The baseline costs obtained for 8, 10, 12, 14, and 16 tasks are 408, 429, 588, 656, and 922, respectively. The number of workers is set to 10. For A-DQN w/ tabu and A-DQN w/ SA, we run A-DQN for 50 episodes, records the weights of the NN, and then apply the local search. The procedure is repeated until the obtained total cost is lower or equal to the baseline cost. It can be seen from Fig. 5(d) that the difference in execution times between the baseline A-DQN and A-DQN with a local search is very large. Both versions of A-DQN with local search obtain a total cost lower than the threshold value within only 50 training episodes of A-DQN in most cases except when the number of tasks is 16. The number of infeasible solutions in the solution space is relatively higher when the task number is 16, and both A-DQN with tabu and SA sometimes need more than 50 episodes to find the desired cost. Therefore, from Fig. 5, we can infer that adding a local search to the DQN algorithm enables obtaining better results in less time. In addition, A-DQN w/ tabu takes slightly less time to obtain the given cost than the A-DQN w/ SA.

3) PERFORMANCE COMPARISON BETWEEN A-DQN W/LOCAL SEARCH AND OTHER MCS ALGORITHMS (i.e., MATC-IGA AND MGATA)

Now, the performance of A-DQN w/ tabu and A-DQN w/ SA is compared with the two other MCS methods (i.e., MGATA and MATC-IGA) under different situations, such as different numbers of tasks, workers, and deadlines.

Fig. 6 shows the total cost obtained from the considered algorithms when the number of tasks varies. From the figure, it is notable that the total cost of all methods (i.e., MGATA, MATC-IGA, A-DQN w/ SA, and A-DQN w/ tabu) gradually increases when the numbers of tasks increase because of the traveling distance and sensing cost. A-DQN w/ tabu and A-DQN w/ SA always outperform MGATA and MATC-IGA, whereas the total cost obtained by the A-DQN-w/ tabu is the lowest among them. For example, when the number of tasks is eight, the total cost obtained by the A-DQN with tabu and SA, respectively is 303.015 and 311.220, whereas MGATA is 394.250. In contrast, MATC-IGA obtains the highest total cost (518.341 when the number of tasks is eight). The results indicate that the DQN-based algorithm can obtain a better solution in the considered problem by approximating the Q-value through NN, which helps the agent learn the policy to find the task–worker assignments with lower costs that maximize the total reward, and locally optimizing the result obtained from the DQN. On the contrary, MGATA and MATC-IGA partially depend on random evolution, and may suffer from loss of genetic diversity resulting in early convergence. In addition, MATC-IGA tries to solve the assignment and worker's path-planning problems directly, and uses a repair operation to obtain a valid solution without any strict rules on the assignment of all tasks. Therefore, solution space is huge, and a lot of invalid solutions are produced in each generation, which makes it hard for MATC-IGA to find a good solution.

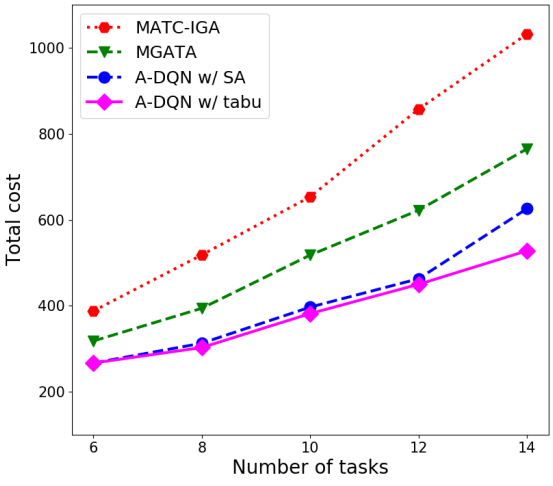

FIGURE 6. The effects of the number of tasks.

Fig. 7 shows the total cost obtained by the four algorithms under the various numbers of workers. It can be seen

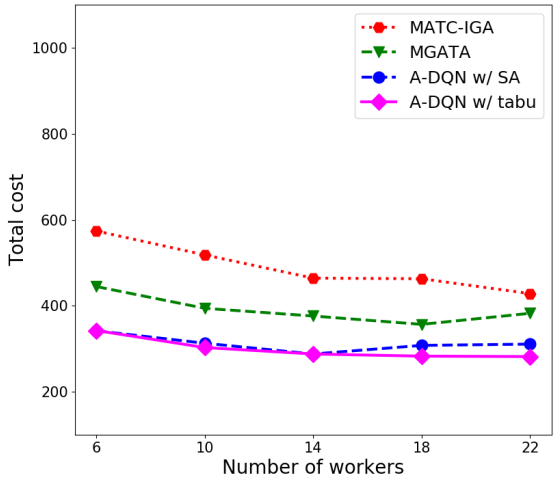

FIGURE 7. The effects of the number of workers.

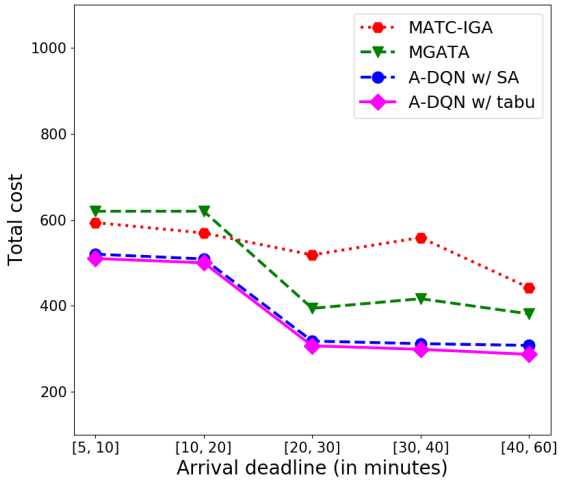

FIGURE 8. The effects of the different deadline (in minutes).

from the figure that a rich worker resource, i.e., a larger number of workers, leads to a lower total cost for all four approaches. This is because a larger group of workers is more diverse; hence, more appropriate workers can be chosen. Furthermore, A-DQN w/ tabu shows the best performance while MATC-IGA being the worst performer. For instance, when the number of workers is 18, the total cost achieved by A-DQN w/ tabu is around 8% lower than A-DQN w/ SA, whereas MGATA and MATC-IGA, respectively, obtain total costs around 19% and 33% higher than A-DQN w/ SA. A-DQN w/ SA achieves the same cost as the A-DQN w/ tabu in some cases (e.g., when the number of workers is 14).

In Fig. 8, the experimental results for different arrival deadlines ranging from [5, 10] to [40, 60] are presented. All four algorithms show a decreasing trend with the increase in the arrival deadline. When the arrival deadline is large, more workers can perform the tasks. For example, distant workers and workers with a lower velocity are able to reach

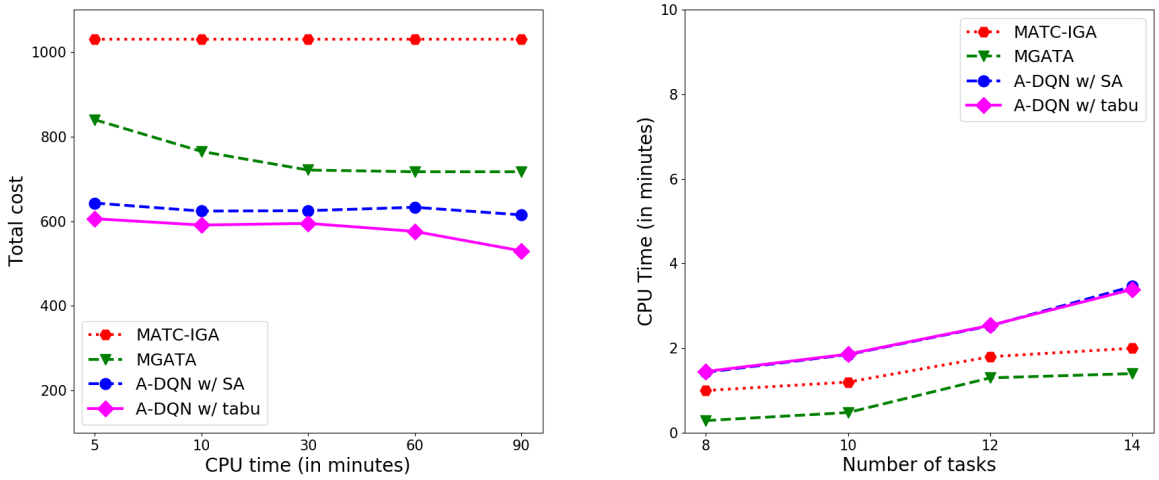

FIGURE 9. The performance of the different algorithms: (a) total cost obtained by each method for different CPU times given 14 tasks and 10 workers, and (b) the time needed by the approaches to obtain the specified total cost under different numbers of tasks.

task venues within the given time. Therefore, the chance of selecting workers at a lower cost increases. It is also shown that there is a slight decrease in total costs of all the algorithms from [5, 10] and [10, 20] as these ranges impose relatively tight deadlines compared to the range [20, 30], where algorithms show a sharp downward trend. For all different ranges of arrival deadlines, A-DQN w/ tabu achieves the lowest total cost. For instance, when the arrival deadline range is between 40 and 60 minutes, A-DQN w/ tabu obtains 287.589 (total cost), which is much lower than MGATA and MATC-IGA.

4) COMPUTATION TIME

The performance of A-DQN w/ tabu, A-DQN w/SA, MGATA, and MATC-IGA after running for a given amount of CPU time, and the CPU time needed by each method to produce the desired result under different situations, are shown in Fig. 9.

First, given different CPU times [5, 10, 30, 60, and 90 minutes], the total cost obtained by each method is shown in Fig. 9(a). In the case of A-DQN w/ tabu and A-DQN w/ SA, A-DQN is trained for 3, 8, 28, 58, and 88 minutes, whereas the local search algorithms are run for two minutes in all cases. Note that the number of iterations in each algorithm is adjusted according to CPU time. Fig. 9(a) shows that the result of MATC-IGA does not improve even though the running time increases from five to 90 minutes, whereas MGATA shows improvement up until 30 minutes. On the other hand, A-DQN with tabu and with SA maintains a decreasing trend until the end, except for a slight increase in A-DQN w/ SA when the CPU time is 60 minutes, possibly because of getting trapped into local minima. A-DQN w/ tabu still achieves the lowest total cost among them. For instance, when the running time is five minutes, A-DQN w/ tabu obtains a total cost of 606.011, whereas A-DQN w/ SA, MGATA, and MATC-IGA achieve 643.423, 840. 051, and 1,030.023, respectively.

Second, how much time each algorithm takes to obtain the specified total cost is shown in Fig. 9(b). The baseline cost is collected by running MATC-IGA because there is no improvement on MATC-IGA after running it for a certain amount of time, as seen in Fig. 9(a). We run MATC-IGA for the different numbers of tasks [8, 10, 12, 14] and obtain respective baseline costs of 518, 653, 857, and 1030. In A-DQN w/ tabu and w/ SA, A-DQN first trained for 10 episodes, and the weights of the NN are recorded; then, the local search is applied. This procedure is repeated until the desired total cost is obtained. It can be seen that DQN-based algorithms take a longer time to obtain the specified cost, whereas MGATA is the fastest. For example, when there are 10 tasks, MGATA and MATC-IGA obtain the desired cost within 0.48 and 1.2 minutes, respectively, whereas A-DQN w/ tabu and w/ SA take 1.85 and 1.86 minutes, respectively. A-DQN w/ SA takes slightly less time than A-DQN w/ tabu in most cases except when the number of tasks is larger, where SA takes a relatively longer time than tabu search.

VII. CONCLUSION

In this paper, we considered a task allocation problem for time-sensitive mobile crowd-sensing applications, where a worker can perform multiple tasks. The problem takes task completion order of the worker (since traveling costs and response times of the tasks assigned to the worker are directly affected by the worker's moving path), as well as the deadline and sensor requirements of tasks, different types of worker (i.e., UAV and human worker), and UAVs' energy constraint into consideration. Specifically, the task allocation problem

has an embedded structure, i.e., it contains multiple task completion order problems, which results in a huge search space. Therefore, first, we formulated the assignment and the task completion order problems as two ILP problems by exploiting a decomposition method where they iteratively interact to achieve the main objective. Next, two variants of the deep Q-learning-based task allocation and worker routing algorithm (A-DQN w/ tabu and A-DQN w/ SA) are proposed to address the assignment problem, where both algorithms employ an ATSP heuristic to solve the traveling salesman problem. Evaluation under various situations demonstrated the effectiveness of the proposed methods in comparison with other recruitment strategies.

REFERENCES

[1] P. M. Santos, C. Queiros, S. Sargento, A. Aguiar, J. Barros, J. G. P. Rodrigues, S. B. Cruz, T. Lourenço, P. M. d'Orey, Y. Luis, C. Rocha, S. Sousa, S. Crisóstomo, and C. Queirós, "PortoLivingLab: An IoT-based sensing platform for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 523–532, Apr. 2018, doi: 10.1109/JIOT.2018.2791522.

[2] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, "Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm," *ACM Comput. Surveys*, vol. 48, no. 1, pp. 1–31, Sep. 2015, doi: 10.1145/2794400.

[3] C. Xu, S. Li, Y. Zhang, E. Miluzzo, and Y.-F. Chen, "Crowdsensing the speaker count in the wild: Implications and applications," *IEEE Commun. Mag.*, vol. 52, no. 10, pp. 92–99, Oct. 2014, doi: 10.1109/MCOM.2014.6917408.

[4] M. Mun, P. Boda, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West, "PEIR, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proc. 7th Int. Conf. Mobile Syst., Appl., Services (Mobisys)*, Kraków, Poland, 2009, pp. 55–68.

[5] R. Gao, M. Zhao, T. Ye, F. Ye, Y. Wang, K. Bian, T. Wang, and X. Li, "Jigsaw: Indoor floor plan reconstruction via mobile crowdsensing," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2014, pp. 249–260.

[6] J. Liu, Y. Li, M. Chen, W. Dong, and D. Jin, "Software-defined Internet of Things for smart urban sensing," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 55–63, Sep. 2015, doi: 10.1109/MCOM.2015.7263373.

[7] Y. Zheng, X. Xie, and W. Y. Ma, "GeoLife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, Jun. 2010.

[8] C. H. Liu, X. Ma, X. Gao, and J. Tang, "Distributed energy-efficient multi-UAV navigation for long-term communication coverage by deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1274–1285, Jun. 2020, doi: 10.1109/TMC.2019.2908171.

[9] H. Xiong, D. Zhang, G. Chen, L. Wang, V. Gauthier, and L. E. Barnes, "ICrowd: Nnear-optimal task allocation for piggyback crowdsensing," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 2010–2022, Aug. 2016, doi: 10.1109/TMC.2015.2483505.

[10] H. Xiong, D. Zhang, G. Chen, L. Wang, and V. Gauthier, "CrowdTasker: Maximizing coverage quality in piggyback crowdsensing under budget constraint," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, St. Louis, MO, USA, Mar. 2015, pp. 55–62.

[11] D. Zhang, H. Xiong, L. Wang, and G. Chen, "CrowdRecruiter: Selecting participants for piggyback crowdsensing under probabilistic coverage constraint," in *Proc. ACM Int. Joint Conf. Pervas. Ubiquitous Comput.*, Sep. 2014, pp. 703–714.

[12] R. Estrada, R. Mizouni, H. Otrok, A. Ouali, and J. Bentahar, "A crowdsensing framework for allocation of time-constrained and location-based tasks," *IEEE Trans. Services Comput.*, vol. 13, no. 5, pp. 769–785, Sep. 2020, doi: 10.1109/TSC.2017.2725835.

[13] M. Abououf, R. Mizouni, S. Singh, H. Otrok, and A. Ouali, "Multi-worker multi-task selection framework in mobile crowd sourcing," *J. Netw. Comput. Appl.*, vol. 130, pp. 52–62, Mar. 2019, doi: 10.1016/j.jnca.2019.01.008.

[14] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, "ActiveCrowd: A framework for optimized multitask allocation in mobile crowdsensing systems," *IEEE Trans. Human-Mach. Syst.*, vol. 47, no. 3, pp. 392–403, Jun. 2017, doi: 10.1109/THMS.2016.2599489.

[15] W. Gong, B. Zhang, and C. Li, "Location-based online task scheduling in mobile crowdsensing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, Dec. 2017, pp. 1–6.

[16] Y. Zhao, Y. Li, Y. Wang, H. Su, and K. Zheng, "Destination-aware task assignment in spatial crowdsourcing," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Singapore, Nov. 2017, pp. 297–306.

[17] X. Li and X. Zhang, "Multi-task allocation under time constraints in mobile crowdsensing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1494–1510, Apr. 2021, doi: 10.1109/TMC.2019.2962457.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[19] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," 2020, *arXiv:2003.03600*. [Online]. Available: http://arxiv.org/abs/2003.03600

[20] S. Akter and S. Yoon, "DaTask: A decomposition-based deadline-aware task assignment and workers' path-planning in mobile crowdsensing," *IEEE Access*, vol. 8, pp. 49920–49932, 2020, doi: 10.1109/ACCESS.2020.2980143.

[21] Y. Wang, H.-X. Li, G. G. Yen, and W. Song, "MOMMOP: Multiobjective optimization for locating multiple optimal solutions of multimodal optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 4, pp. 830–843, Apr. 2015, doi: 10.1109/TCYB.2014.2337117.

[22] G. Pataki, "Teaching integer programming formulations using the traveling salesman problem," *SIAM Rev.*, vol. 45, no. 1, pp. 116–123, Jan. 2003, doi: 10.1137/S00361445023685.

[23] F. S. He, Y. Liu, A. G. Schwing, and J. Peng, "Learning to play in a day: Faster deep reinforcement learning by optimality tightening," Nov. 2016, *arXiv:1611.01606*. [Online]. Available: http://arxiv.org/abs/1611.01606

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[25] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992, doi: 10.1007/BF00992698.

[26] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988, doi: 10.1007/BF00115009.

[27] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with a natural language action space," in *Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL)*, Berlin, Germany, Aug. 2016, pp. 1621–1630.

[28] M. A. Arostegui, S. N. Kadipasaoglu, and B. M. Khumawala, "An empirical comparison of tabu search, simulated annealing, and genetic algorithms for facilities location problems," *Int. J. Prod. Econ.*, vol. 103, no. 2, pp. 742–754, Oct. 2006, doi: 10.1016/j.ijpe.2005.08.010.

[29] M. S. Hussin and T. Stützle, "Tabu search vs. simulated annealing as a function of the size of quadratic assignment problem instances," *Comput. Oper. Res.*, vol. 43, pp. 286–291, Mar. 2014, doi: 10.1016/j.cor.2013.10.007.

[30] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Springer, 1998, pp. 2093–2229.

[31] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983, doi: 10.1126/science.220.4598.671.

[32] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang, "The asymmetric traveling salesman problem: Algorithms, instance generators, and tests," in *Algorithm Engineering and Experimentation* (Lecture Notes in Computer Science), vol. 2153. Berlin, Germany: Springer, 2001, doi: 10.1007/3-540-44808-X-3.

[33] L. Bracciale. (Jul. 2014). *CRAWDAD Dataset Roma/Taxi, Version 2014-07-17*. [Online]. Available: https://crawdad.org/roma/taxi/20140717

SHATHEE AKTER received the M.Sc. degree in computer engineering from the University of Ulsan, South Korea, in 2020, where she is currently pursuing the Ph.D. degree. Her current research interests include mobile crowd-sensing, optimization algorithms, and the Internet of Things.

THI-NGA DAO received the B.S. degree in electrical and communication engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Ulsan, South Korea, in 2016 and 2019, respectively. She is currently a Lecturer with the Faculty of Radio Engineering, Le Quy Don Technical University. Her research interests include machine learning-based applications and optimization problems in mobile crowd-sensing, human mobility, and network security.

SEOKHOON YOON (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science and engineering from the State University of New York at Buffalo (SUNY Buffalo), in 2005 and 2009, respectively. After receiving the Ph.D. degree, he worked as a Principal Research Engineer with Defense Industry, where he designed several tactical wireless network solutions. He is currently an Associate Professor with the University of Ulsan, South Korea, where he leads the Advanced Mobile Networks Laboratory. His research interests include opportunistic networks, optimization algorithms, machine learning-based IoT service, intelligence defined networking, and mobile crowd-sensing.