DeepSensing: A Novel Mobile Crowdsensing Framework with Double Deep Q-Network and Prioritized Experience Replay

Xi Tao and Abdelhakim Senhaji Hafid, *Member, IEEE*

***Abstract*—Mobile crowdsensing (MCS) is a new and promising paradigm of data collection due to the growing number of mobile smart devices. It can be utilized in applications of large-scale sensing by employing a group of mobile users with their smart devices. Since a large number of mobile users are recruited, the allocation of sensing tasks to mobile users has a critical influence on the performance of MCS applications. To efficiently assign sensing tasks to mobile users, we propose a novel MCS framework named DeepSensing. This framework consists of six executive phases, *i.e.*, registration of sensing tasks, announcement of reward rule, collection of users' information, task allocation, execution of sensing activities, and distribution of data and rewards. Here, the phase of task allocation is a key component, which directly determines the performance of DeepSensing, *e.g.*, the platform's profit. DeepSensing aims to maximize the platform's profit by taking into account the various constraints of sensing tasks and mobile users. Therefore, we propose a deep reinforcement learning (DRL) method to optimally assign sensing tasks to mobile users. Specifically, we employ a double deep Q-network with prioritized experience replay (DDQN-PER) to address the task allocation problem, which is also formulated as a path planning problem with time windows. To evaluate our proposed DDQN-PER solution, three baseline solutions are provided, *i.e.*, ant colony system (ACS), ϵ-greedy, and random solutions. Finally, the results of numerical simulations show that our proposed DDQN-PER solution outperforms the baseline solutions in terms of the platform's profit and it plans better-organized travelling paths for mobile users.**

***Index Terms*—Mobile crowdsensing, task allocation, deep reinforcement learning, double deep Q-network, prioritized experience replay.**

I. Introduction

MOBILE crowdsensing enables large-scale sensing by recruiting a large number of mobile users with their smart devices (*e.g.*, smart phones, laptops, and wearables) or vehicles with embedded sensors (*e.g.*, autonomous cars and unmanned aerial vehicles). The term *mobile crowdsensing* was coined by Ganti *et al.* [1] and it has already gained significant attention because of its applicability in data collection [2]. MCS can provide a flexible and sufficient coverage of sensing tasks or areas using the mobility of mobile users. In addition, the human intelligence of mobile users can be used to accomplish complex sensing tasks. For instance, mobile users can easily identify a bird and take a picture of it with their smart phones when they are involved in a bird monitoring application. MCS is also a cost-efficient method of data collection when it is compared with conventional wireless sensor networks. It collects data directly from mobile users without infrastructure cost (*e.g.*, networks and equipment) and the corresponding maintenance cost. Because of these advantages, MCS is widely used in a variety of sensing and computing applications, *e.g.*, environment monitoring [3], transportation management [4], and healthcare [5], to name a few.

Although MCS benefits from user participation, the participation of numerous mobile users does pose many challenges. First, it is hard to guarantee the quality of collected data or information [6]. Various types of employed devices may lead to a fluctuation of data quality. Furthermore, the skills or knowledge of mobile users also have a influence on data quality. The second challenge concerns the motivation of user participation, which is also known as the problem of incentive mechanism design [7,8]. As such, an efficient method of determining rewards or payments to mobile users is critical in a budget-limited MCS application. Finally, the task allocation problem is another big challenge because of various requirements and constraints of sensing tasks and mobile users. Sensing tasks should be carefully assigned to the appropriate mobile users in order to achieve some specific targets, *e.g.*, energy efficiency.

With respect to the task allocation problem, the assignment mode should be determined first. Generally, there are two different ways to assign sensing tasks, *i.e.*, platform-centric and user-centric modes [9,10]. In the platform-centric mode, the centralized platform has all of the information from sensing tasks and mobile users. As a result, global optimization algorithms can be employed to determine the task set assigned to each mobile user. In the user-centric mode, mobile users select sensing tasks by themselves and then they submit their task sets to the platform. The platform only needs to make a selection among the submitted task sets. Although mobile users have the autonomy to choose sensing tasks, there are several drawbacks in the user-centric mode, *e.g.*, competition and computation issues. The competition among mobile users makes the sensing tasks with higher rewards much more popular. That is to say, mobile users have to compete with each other to win the high-reward tasks. As a consequence, this competition issue leads to an unbalanced assignment of sensing tasks. Then, the computation issue means that it is computationally intractable for mobile users to find their optimal task sets with their resource-limited devices.

This research was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada.

X. Tao and A. Hafid are with the Department of Computer Science and Operations Research, University of Montreal, Montreal, QC H3T 1J4, Canada (e-mails: xi.tao@umontreal.ca, ahafid@iro.umontreal.ca).

In MCS applications, sensing tasks are generally located at specific points or areas [11]. Therefore, mobile users have to reach these locations to perform their assigned tasks. Since the order of performing the tasks is significantly important for each mobile user [12], the task allocation problem can be considered as a path planning problem. However, the path planning problem is computationally intractable even with only one mobile user [10]. To plan travelling paths for mobile users, the platform makes decisions regarding sensing tasks to mobile users in a sequential manner. From the perspective of decision making, reinforcement learning (RL) is known as a powerful tool to solve such a problem [13]. Then, the platform is regarded as the agent of RL. The status of sensing tasks and mobile users represents the environment of RL. Thus, RL is promising when applied to the task allocation problem. However, conventional RL methods (*e.g.*, Q-learning [14]) suffer from a slow convergence speed and a request for big data storage when the state and action spaces are huge. To overcome this issue, deep reinforcement learning (DRL) introduces deep learning methods into RL, which has already achieved great improvements in playing video games [15] and Go [16].

In this paper, we propose a novel framework of MCS, *i.e.*, DeepSensing, which is based on the platform-centric mode. There are six executive phases of DeepSensing. At the very beginning, data requesters publish their sensing tasks with the associated information (*e.g.*, locations, time windows and budgets) in the platform. Second, the platform needs to determine and announce a reward rule to mobile users. Third, mobile users register in the platform if they accept the reward rule and then provide their private information (*e.g.*, participation time) to the platform. Next, the platform solves the task allocation problem and plans travelling paths for the recruited mobile users. In the fifth phase, the recruited mobile users move along the planned paths to perform their assigned tasks and then upload the collected data to the platform. The last step is that the platform verifies the collected data and sends the data to the corresponding data requesters. Meanwhile, the platform distributes rewards to the recruited mobile users. In existing studies, the platform and data requesters are frequently considered to have consistent interests, *e.g.*, data quality and sensing cost. However, the top priority of a commercial platform is its profit. Thus, the objective of DeepSensing is to maximize the platform's profit. To achieve this objective, we propose a solution, *i.e.*, double deep Q-network with prioritized experience replay (DDQN-PER), to the task allocation problem. In addition, we consider three baseline solutions, *i.e.*, ant colony system (ACS), ϵ-greedy, and random solutions, for performance comparison. Finally, we evaluate our proposal with the baseline solutions via simulations.

The contributions of our paper can be summarized as follows:

- We propose a novel framework of MCS, *i.e.*, DeepSensing, as a complete system to efficiently recruit mobile users and accomplish sensing tasks. DeepSensing efficiently plans travelling paths for mobile users to perform their assigned sensing tasks and achieve specific targets.
- We propose a deep reinforcement learning method to solve the task allocation problem in DeepSensing. Specifically, we are the first to use a double deep Q-network with prioritized experience replay to address the task allocation problem of MCS.
- We propose three baseline solutions (*i.e.*, ant colony system, ϵ-greedy, and random solutions) and conduct various simulations with different numbers of mobile users to evaluate the performance of DeepSensing. The results show that our proposed DDQN-PER solution outperforms the baseline solutions, in terms of the platform's profit, and plans better-organized travelling paths for mobile users.

The remainder of this paper is organized as follows. Section II presents related work. Section III describes the structure of DeepSensing and the formulation of the task allocation problem. Section IV presents details of DDQN-PER solution and three baseline solutions. Section V evaluates our proposal via simulations. Finally, Section VI concludes the paper.

II. Related Work

In this section, we overview related work with respect to task allocation of MCS and deep reinforcement learning in networking.

A. Task Allocation of MCS

The task allocation problem of MCS has already been studied from many perspectives [17]–[19]. Existing solutions focus on two metrics: data quality and sensing cost. Although data quality has different definitions depending on various requirements in MCS applications, it is commonly defined as the coverage of sensing tasks or areas. Zhang *et al.* [20] proposed a user selection framework, called CrowdRecruiter, where a probabilistic coverage is used. The probabilistic coverage is defined as the ratio of covered cell towers in a time frame. CrowdRecruiter leverages historical records of mobile users to predict their phone calls. Then, sensing tasks are assigned based on the prediction. A large-scale real-world dataset is used to evaluate the performance of CrowdRecruiter and the results did show that it can recruit a small number of mobile users to satisfy the coverage requirement. Xiong *et al.* [21] proposed a generic MCS task allocation framework, called iCrowd. In iCrowd, a spatial-temporal coverage, called k-depth coverage, is used. k-depth coverage is defined as the maximum number of desired data samples in one subarea. iCrowd also leverages historical records of mobile users to predict their mobility and hence efficiently recruit them. To address the online recruitment problem, Liu *et al.* [22] proposed a dynamic user recruitment strategy to maximize the expected number of completed tasks under the budget and time constraints. In addition to the coverage, data quality is alternatively defined as a value or utility, which is related to the total number of data samples [10,23]. Although data quality improves with the increasing number of data samples, the total number of data samples is limited by a budget of sensing tasks. For instance, He *et al.* [23] assumed a maximum number of data samples for each sensing task and we also considered a diminishing

marginal increase of data quality [10] to meet a requirement of the budget.

In addition to data quality, sensing cost is another aspect of interest in MCS applications. Generally, sensing cost is considered as energy cost. To reduce energy consumption of task allocation, several energy-efficient paradigms have been proposed, *e.g.*, piggyback crowdsensing (PCS) and compressive crowdsensing (CCS). PCS leverages opportunities of phone activities to collect data with low-energy cost [24]. This paradigm has been used by a number of MCS frameworks to achieve energy efficiency [20,21,25,26]. In EMC^3 [25] and EEMC [26], the energy consumption of data transfer is reduced by leveraging phone calls of mobile users. CCS employs compressive sensing [27,28] to reconstruct a global data distribution from sparsely collected data and further reduce energy cost. For instance, CCS is used to minimize the number of sensed subareas in each sensing cycle and infer the data of unvisited subareas to meet a requirement of probabilistic data accuracy [29]. Liu *et al.* [30] proposed a CCS-based recovery scheme, called UniTask, to improve the overall utility of MCS applications, *e.g.*, coverage, latency, and accuracy. In MCS applications, travelling distance is another cost metric used [10,23,31]. Guo *et al.* [31] proposed a worker selection framework, called ActiveCrowd. It takes into account travelling distance of each mobile user and two types of sensing tasks (*i.e.*, time-sensitive and delay-tolerant tasks) in the process of task allocation. In addition, data uploading cost is also important in MCS applications. Wang *et al.* [32] proposed an efficient prediction-based user recruitment strategy to reduce uploading cost by relaying uploaded data among mobile users.

In existing studies, data quality and sensing cost are considered from the perspective of mobile users. However, the platform's concern is omitted. In commercial MCS applications, the platform's objective is to make a profit by its intermediary services. Therefore, in this paper, we consider the task allocation problem of MCS from the platform's perspective and aim to maximize its profit. In addition, the relationship between travelling time and distance is not clearly defined in existing studies. For instance, although a maximum travelling distance is proposed for the path of each mobile user in [23], the travelling time along the path is not known. This is a serious issue when time-sensitive tasks are considered in MCS applications. To clarify the relationship between travelling time and distance, we define a travelling speed for each mobile user and we also consider time windows of sensing tasks.

B. Deep Reinforcement Learning in Networking

Reinforcement learning enables the learning process of an agent by interacting with its environment [13]. The agent can improve its decisions during the learning process. However, the learning process has a slow convergence speed when the state and action spaces are huge. To tackle this issue, deep learning [33] is leveraged to enhance reinforcement learning as deep reinforcement learning (DRL) [34,35]. DRL has been already applied in many areas [15,16]. In DRL methods, deep Q-network (DQN) is widely used to approximate Q-values and it replaces the Q-value table in conventional RL methods. To further improve the performance of DQN, various techniques are proposed, *e.g.*, double deep Q-network [36] and prioritized experience replay [37].

Recently, DRL has been applied in the area of networking as an emerging tool to address some problems and challenges [38,39]. In [39], Xiong *et al.* illustrated the potential of DRL to be an efficient solution to many networking problems, *e.g.*, computation offloading, edge caching, and network slicing. In particular, they proposed a DRL-based scheme for network slicing. The scheme is then compared, via simulations, with the baseline schemes (*e.g.*, conventional Q-learning, greedy, and random schemes). The results show that the DRL-based scheme outperforms the conventional communication resource allocation schemes. DRL is also used in vehicular networks to solve a joint edge computing and caching problem to maximize system utility [40]. In addition, Dai *et al.* [41] integrated DRL with blockchain to provide a secure and intelligent resource sharing system. They presented an example of content caching system. First, they used a consortium blockchain to create a secure environment and then they leveraged DRL to address the content caching problem to maximize the utility of caching resources.

Since DRL has the ability to solve optimization problems in the area of networking, we desire to leverage this emerging method to solve the task allocation problem of MCS [42]. As far as we know, we are the first to solve the task allocation problem of MCS by using a double deep Q-netwrok with prioritized experience replay. We propose to use deep neural networks (DNNs) in RL for function approximation, because conventional RL methods (*e.g.*, Q-learning, Monte Carlo method, and dynamic programming) cannot satisfy the requirements of computation and storage when state and action spaces are huge. In addition, we employ double Q-learning method to achieve an unbiased action estimation during the training process. We also use prioritized experience replay to accelerate the convergence of the training process. In conclusion, our proposed method, *i.e.*, double deep Q-network with prioritized experience replay, performs better in terms of efficiency and convergence when it is compared with conventional RL methods.

III. Framework Structure and Problem Formulation

In this section, we present the structure of DeepSensing. Then, we formulate the task allocation problem of MCS as a path planning problem, and then analyze its complexity. Table I shows notations that are used in the rest of the paper.

A. Structure of DeepSensing

There are six executive phases in DeepSening, *i.e.*, registration of sensing tasks, announcement of reward rule, collection of users' information, task allocation, execution of sensing activities, and distribution of data and rewards. Fig. 1 shows the structure of DeepSensing.

TABLE I
NOTATION DEFINITIONS.

Notations	Definitions
V	Set of sensing tasks
V'	Set of assigned tasks
B	Set of tasks' budgets
$[g_i^s, g_i^e]$	Time window of task v_i
$t(v_i)$	Complete time of task v_i
W	Set of mobile users
W'	Set of recruited users
q_j	Reward to user w_j
P_j	Travelling path of user w_j
$d(P_j)$	Travelling distance of path P_j
λ_j	Reward per distance unit of user w_j
$[h_j^s, h_j^e]$	Participation time of user w_j
f_j	Travelling speed of user w_j
$\tau(P_j)$	Waiting time of user w_j along path P_j
u	Platform's Profit

1) Registration of Sensing Tasks: During the first step, data requesters publish their sensing tasks in the platform and deposit the associated budgets to the platform as rewards to mobile users. Meanwhile, detailed information of the sensing tasks, *e.g.*, locations, is sent to the platform. The set of the sensing tasks is denoted by $V = \{v_1, v_2, ..., v_m\}$ and the associated set of their budgets is represented by $B = \{b_1, b_2, ..., b_m\}$. We assume that each sensing task $v_i \in V$ is time-sensitive and it can only be accomplished within a time window $[g_i^s, g_i^e]$.

2) Announcement of Reward Rule: Once receiving the information of the sensing tasks, the platform announces a reward rule to mobile users. The reward rule is a method to determine the amount of the reward to each mobile user. The set of mobile users is represented by $W = \{w_1, w_2, ..., w_n\}$. For each mobile user $w_j \in W$, the reward is determined by

$$q_j = \lambda_j \cdot d(P_j) \tag{1}$$

where P_j is the planned path for mobile user w_j. $d(P_j)$ is a function to measure the travelling distance of path P_j and λ_j is the reward per distance unit that mobile user w_j desires. It is worth noting that different reward rules can be used in MCS applications according to their requirements. For instance, energy cost can be involved in the reward rule with computationally intensive sensing tasks. In this paper, energy cost is not considered in the reward rule because we only have light-weighted sensing tasks with little energy consumption, *e.g.*, photo taking. However, our proposed framework can be easily adapted to work with different reward rules.

3) Collection of Users' Information: Mobile users need to make decisions to whether to participate in upcoming sensing activities based on the reward rule and their status. If they are willing to participate, they provide their private information (*e.g.*, participation time and travelling speed) to the platform. For each mobile user $w_j \in W$, the participation time is denoted by $[h_j^s, h_j^e]$ and the travelling speed is denoted by f_j. Here, the travelling speed is strongly influenced by the method of travel, *e.g.*, walking, cycling, and driving. In addition, mobile user w_j needs to inform the platform of the initial location and the desired reward per distance unit (*i.e.*, λ_j).

4) Task Allocation: In DeepSensing, the platform acts as an intermediary between data requesters and mobile users. It earns a profit from its brokerage service by assigning sensing tasks to mobile users. The platform's profit is the difference between the total budget of the sensing tasks deposited by their data requesters and the total reward to the mobile users. We define the platform's profit u as follows:

$$u = \sum_{w_j \in W'} \Big(\sum_{v_i \in P_j} b_i - q_j \Big) \tag{2}$$

where $\sum_{v_i \in P_j} b_i$ is the total budget of the selected sensing tasks along path P_j and q_j is the corresponding reward to mobile user w_j. W' represents the set of the recruited mobile users.

To tackle the path planning problem, *i.e.*, the task allocation problem of MCS, the platform needs to carefully plan paths for the recruited mobile users. The objective is to maximize the platform's profit. Currently, there are three constraints that need to be considered. The first one is that mobile users can only perform their assigned sensing tasks within their participation time. Second, their assigned tasks can only be performed within the associated time windows. Finally, each sensing task can only be assigned to at most one mobile user. According to the objective and constraints, the task allocation problem is formulated as follows:

$$\max.\ u \tag{3a}$$

$$\text{s.t.}\quad h_j^s + \frac{d(P_j)}{f_j} + \tau(P_j) \le h_j^e, \forall w_j \in W' \tag{3b}$$

$$g_i^s \le t(v_i) \le g_i^e, \forall v_i \in V' \tag{3c}$$

$$\sum_{w_j \in W'} 1(v_i \in P_j) \le 1, \forall v_i \in V'. \tag{3d}$$

Here, the objective function is defined in (3a). Constraint (3b) indicates that each recruited mobile user $w_j \in W'$ must finish the planned path before the end of the participation time, where $\frac{d(P_j)}{f_j}$ is the travelling time of mobile user w_j along path P_j and $\tau(P_j)$ is the waiting time of mobile user w_j along path P_j. The waiting time arises when mobile users arrive at the locations of the assigned sensing tasks before the starts of the time windows. It is worth mentioning that we omit the time of performing sensing tasks, because it does not take a long time to accomplish the light-weighted tasks that are considered in

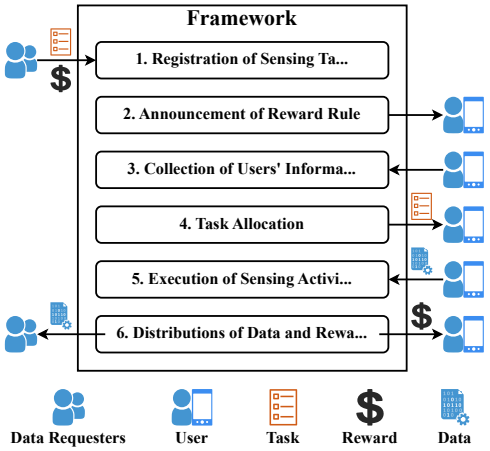


Fig. 1. Structure of DeepSensing.

DeepSensing. For instance, it may only take seconds to send a message or take a photo by smartphones. Constraint (3c) guarantees that each assigned sensing task $v_i \in V'$ can be performed within the corresponding time window, where V' is the set of the assigned sensing tasks and $t(v_i)$ is the time when sensing task v_i is finished. At last, constraint (3d) indicates that each assigned sensing task $v_i \in V'$ is performed by at most one recruited mobile user, where $1(\cdot)$ is an indicator function and it equals 1 when the condition in the function argument is satisfied.

5) Execution of Sensing Activities: In this phase, the recruited mobile users move along their planned paths to perform the assigned sensing tasks. Once they finish the paths, they upload the collected data to the platform, which can apply certain techniques (*e.g.*, machine learning methods) to verify and evaluate the uploaded data. Then, a reputation system can be built to rank mobile users based on the quality of their collected data. This reputation is a good indicator for future task allocation.

6) Distribution of Data and Rewards: This is the last phase of the current sensing cycle. The platform sends the collected data to data requesters and distributes rewards to the recruited mobile users. Then, the platform starts the next cycle of sensing activities.

B. Computational Hardness of Task Allocation Problem

Next, we analyze the computational complexity of the formulated task allocation problem.

Theorem 1. *The task allocation problem in* (3) *is NP-hard.*

Proof. We reduce a known NP-hard problem, *i.e.*, the orienteering problem [43], to an instance of our formulated problem in (3). The orienteering problem is defined in a game context. A player in this game needs to plan a tour to visit a number of tasks that are associated with certain locations and game points. The objective is to maximize the total points collected by the player with a limit on the travelling distance. This game is also called the orienteering problem with time windows, when the player can only perform a task within a specific time period to gain points [44]. The orienteering problem (with time windows) is already proved to be NP-hard [43,44].

Let us construct an instance of the task allocation problem to solve the orienteering problem. We assume that there is only one mobile user in this scenario, who acts as the player in the orienteering problem. This mobile user has a predetermined travelling speed and participation time. That is to say, the mobile user has a maximum travelling distance, which is mapped to the travelling distance limit in the orienteering problem. In our instance, the time windows of all sensing tasks are open during the user's tour. The reward to the user for accomplishing the assigned tasks is set to 0. As a result, the platform's profit equals the total budget of the finished tasks according to (2) and it is mapped to the final game points in the orienteering problem. Hence, this constructed instance of the task allocation problem is, in fact, an orienteering problem. We conclude that the task allocation problem in (3) generalizes the orienteering problem and it is also NP-hard. □

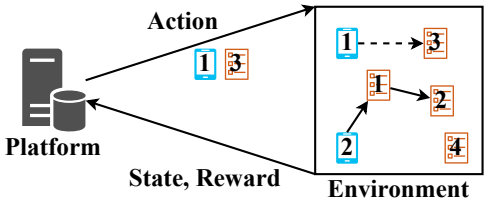


Fig. 2. Task allocation as reinforcement learning.

IV. Solutions to Task Allocation Problem

In this section, we first consider the task allocation problem of MCS from the perspective of reinforcement learning (RL) and then we propose a solution based on double deep Q-network with prioritized experience replay (DDQN-PER). To evaluate the proposed solution, we also provide three baseline solutions as benchmarks, including ant colony system (ACS), ϵ-greedy, and random solutions.

A. DDQN-PER Solution

From the perspective of RL, the platform is regarded as the agent and the environment is represented by the status of sensing tasks and mobile users (see Fig. 2). Then, the task allocation problem can be considered as a Markov decision process (MDP) and represented by a tuple (S, A, Z, R). S is a finite set of states and each state is simply defined by paths of mobile users at the current moment. The other variables of the environment (*e.g.*, assignments of sensing tasks, current locations of mobile users, and travelling distances of mobile users) can be calculated based on the current paths of mobile users and the initial state of the environment. A is a finite set of actions and each action is an assignment between one sensing task and one mobile user. Z is the transition function, which turns the current state s to the next state s' with the selected action a, *i.e.*, $s' = Z(s, a)$. R is the reward function and it returns the reward of the selected action under the current state, which is also the platform's profit obtained from the selected assignment. Then, the task allocation problem can be solved by finding the optimal policy π for the platform, *i.e.*, $a = \pi(s)$.

RL problems can be addressed in various ways, *e.g.*, dynamic programming, Monte Carlo methods, and temporal difference methods. The temporal difference methods are widely used, given that they are model-free and learn directly from unfinished episodes. Q-learning is a representative of temporal difference methods with a Q-value table to record the quality of each state-action pair, *i.e.*, $Q : S \times A \to \mathbb{R}$. For state $s_t \in S$ and action $a_t \in A$ at the step t, the Q-value of this state-action pair is estimated by

$$\begin{aligned} Q^{new}(s_t, a_t) \leftarrow & (1-\alpha) \cdot Q(s_t, a_t) \\ & + \alpha \cdot \left(r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right) \end{aligned} \quad (4)$$

where α is the learning rate. γ is the discount factor and r_t is the reward obtained when the state moves from s_t to s_{t+1} by taking action a_t.

However, if state space S and action space A are huge, the conventional RL methods become infeasible due to the steep

requirement of computation and storage. To address this issue, deep Q-network introduces deep neural networks to effectively approximate Q-values. Given state s_t, DQN outputs a vector of action values $Q(s_t, \cdot; \theta)$, where θ are parameters of a DNN (*i.e.*, policy network). In DQN, two important concepts are used, *i.e.*, target network and experience replay. The target network, with parameters θ^-, has the same structure as the policy network. Its parameters are copied from the policy network every C steps, and kept constant at other steps. With respect to the experience replay, there is a memory bank to store observed transitions during the learning process. The stored transitions are uniformly sampled at each step to update the policy network. Then, the Q-value in DQN is updated by

$$Q^{new}(s_t, a_t; \theta) \leftarrow (1-\alpha) \cdot Q(s_t, a_t; \theta) + \alpha \cdot \Big(r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \theta^-)\Big). \quad (5)$$

The max operator in DQN selects overestimated values and results in overoptimistic value estimates. To decouple the selection from the evaluation, double deep Q-network is proposed in [36]. In DDQN, the policy network is used to greedily select actions, and the target network is used to estimate their values. DDQN updates the Q-value by

$$Q^{new}(s_t, a_t; \theta) \leftarrow (1-\alpha) \cdot Q(s_t, a_t; \theta) + \alpha \cdot \Big(r_t + \gamma \cdot Q\big(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \theta^-\big)\Big). \quad (6)$$

In DQN, the experience replay is used to stabilize the training of DNNs. The transitions in replay memory are sampled uniformly at random. However, the sampled transitions are not equally important to the learning process. Therefore, prioritized experience replay (PER) is proposed in [37] and it sets a priority for each transition according to temporal-difference (TD) error. The TD error can indicate how unexpected a transition is. The transitions with larger TD error are more likely to be selected from replay memory during the learning process. As a result, the platform learns more efficiently from the sampled transitions. For each transition $z_i \in Z'$, the TD error is represented by δ_i. Here, the set of the stored transitions is denoted by Z'. The priority of transition z_i is computed by

$$x_i = |\delta_i| + \zeta \quad (7)$$

where ζ is a small positive constant to guarantee that each transition can be sampled even with the TD error of zero.

Next, a stochastic sampling method is leveraged in the prioritized experience replay. The selection probability of a transition is monotonic with the priority. Specifically, the probability of sampling transition z_i is calculated by

$$X(z_i) = \frac{x_i^\beta}{\sum_{z_k \in Z} x_k^\beta} \quad (8)$$

where β indicates how much prioritization is used. It is worth mentioning that a novel data structure named SumTree is used to implement the replay memory [37]. It is a binary tree, where the value of each node is equal to the sum of values associated with the nodes in its left and right subtrees.

Algorithm 1: DDQN-PER solution.

Input: V (tasks), W (users), N (capacity of replay memory), ϵ (probability of random selection), M (maximum number of episodes with no improvement), γ (discount factor), C (frequency of updating target network)

Output: u (platform's profit), $\{P_j : \forall w_j \in W\}$ (travelling paths of mobile users)

Initialize policy network Q with parameters θ
Initialize target network $\hat{Q}$ with parameters $\theta^- = \theta$
Initialize replay memory D with capacity of N
$u \leftarrow -\infty$ *// global maximum profit*
$k \leftarrow 0$ *// number of episodes with no improvement*
while $k < M$ **do**
  $u_k \leftarrow 0$ *// local profit in current episode*
  $s_t \leftarrow s_0$ *// initialize state*
  while $s_t \neq s_e$ **do** *// s_t is not terminal state*
    Randomly select a_t with probability ϵ; otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute a_t to observe r_t and s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in D
    Sample a minibatch of transitions D' from D according to (7) and (8)
    for $\forall z_i = (s_i, a_i, r_i, s_{i+1}) \in D'$ **do**
      if $s_{i+1} \neq s_e$ **then** *// s_{i+1} is not terminal state*
        $y_i = r_i + \gamma \cdot Q\big(s_{i+1}, \arg\max_a Q(s_{i+1}, a; \theta); \theta^-\big)$
      else
        $y_i = r_i$
    Perform gradient descent with respect to θ
    Compute and record TD error $\{\delta_i : \forall z_i \in D'\}$
    Reset $\hat{Q} \leftarrow Q$ every C steps
    $s_t \leftarrow s_{t+1}$ *// go to next step*
    $u_k = u_k + r_t$ *// update local profit*
    Update status of tasks and users
  if $u_k > u$ **then**
    $u = u_k$ *// update global maximum profit*
    Record paths $\{P_j : \forall w_j \in W\}$
    $k = 0$ *// reset k with improvement*
  else
    $k = k + 1$ *// increase k without improvement*
return u and $\{P_j : \forall w_j \in W\}$

With respect to the probability of action selection at each step, it is a good strategy to use a large one at the beginning to widely explore the solution space and then gradually reduce to a small probability at the end, in order to exploit the best policy that has been found. In this case, the probability of random selection ϵ starts with a large value ϵ^{max} and then decays exponentially towards a small value ϵ^{min}. A decay factor η controls the decay rate. Then, ϵ is determined as follows:

$$\epsilon = \epsilon^{min} + (\epsilon^{max} - \epsilon^{min}) * e^{-k/\eta} \quad (9)$$

where k indicates the current episode.

Algorithm 2: ACS solution.

Input: V (tasks), W (users), N (number of ants), ϵ (probability of random selection), M (maximum number of episodes with no improvement)
Output: u (platform's profit), $\{P_j : \forall w_j \in W\}$ (travelling paths of mobile users)

1 $u \leftarrow -\infty$ *// global maximum profit*
2 $k \leftarrow 0$ *// number of episodes with no improvement*
3 Initialize pheromone
4 **while** $k < M$ **do**
5 $u_k \leftarrow -\infty$ *// local profit in current episode*
6 **for** p *from* 1 *to* N **do** *// for each ant*
7 $u_p \leftarrow 0$ *// profit of ant*
8 **for** $\forall v_i \in V$ **do** *// for each task*
9 **if** *there exist feasible users for task* v_i **then**
10 Random selection with probability ϵ; otherwise, greedy selection (pheromone)
11 Get profit of this assignment as u_i
12 $u_p \leftarrow u_p + u_i$ *// update profit of ant*
13 Update status of tasks and users
14 Locally update pheromone
15 **if** $u_p > u_k$ **then**
16 $u_k = u_p$ *// update local profit*
17 Record paths $\{P_j : \forall w_j \in W\}$ of current ant
18 Globally update pheromone
19 **if** $u_k > u$ **then**
20 $u = u_k$ *// update global maximum profit*
21 Record paths $\{P_j : \forall w_j \in W\}$ in current episode
22 $k = 0$ *// reset k with improvement*
23 **else**
24 $k = k + 1$ *// increase k without improvement*
25 **return** u and $\{P_j : \forall w_j \in W\}$

Algorithm 3: ϵ-greedy solution.

Input: V (tasks), W (users), ϵ (probability of random selection), M (maximum number of episodes with no improvement)
Output: u (platform's profit), $\{P_i : \forall w_i \in W\}$ (travelling paths of mobile users)

1 $u \leftarrow -\infty$ *// global maximum profit*
2 $k \leftarrow 0$ *// number of episodes with no improvement*
3 **while** $k < M$ **do**
4 $u_k \leftarrow 0$ *// local profit in current episode*
5 **for** $\forall v_i \in V$ **do** *// for each task*
6 **if** *there exist feasible users for task* v_j **then**
7 Random selection with probability ϵ; otherwise, greedy selection (profit)
8 Get profit of this assignment as u_i
9 $u_k \leftarrow u_k + u_i$ *// update local profit*
10 Update status of tasks and users
11 **if** $u_k > u$ **then**
12 $u = u_k$ *// update global maximum profit*
13 Record paths $\{P_j : \forall w_j \in W\}$
14 $k = 0$ *// reset k with improvement*
15 **else**
16 $k = k + 1$ *// increase k without improvement*
17 **return** u and $\{P_j : \forall w_j \in W\}$

Based on DDQN in (6), PER in (7) and (8), and decayed exploration in (9), we propose a DDQN-PER solution to the task allocation problem in DeepSensing. Alg. 1 shows details of the DDQN-PER solution. Lines 1-3 initialize policy network, target network, and replay memory, respectively. The replay memory is a cyclic buffer to record transitions. Line 4 sets a variable to record the global maximum profit. The learning process ends when it cannot improve the global maximum profit within a certain number of continuous episodes. A variable (*i.e.*, k) is used to count episodes with no improvement in line 5. Line 6 indicates that the learning process continues if k does not reach a predefined threshold (*i.e.*, M). In each episode, a local profit and start state are initialized first in lines 7-8. If the current state is not a terminal state (line 9), a ϵ-greedy strategy is applied to select an action based on the policy network (line 10). In the ϵ-greedy strategy, a random action is selected with probability ϵ; Otherwise, the platform greedily selects the action with the largest Q-value. By executing the selected action, the reward and next state are observed by the platform (line 11). Then, the complete transition is also stored in the replay memory (line 12). During each learning step, the platform samples a minibatch of transitions (line 13) to update the policy network according to (6) (lines 14-18). Next, a gradient descent step on $\sum_{z_i \in D'} \left(y_i - Q(s_i, a_i; \theta)\right)^2$ is performed in line 19 with respect to parameters θ. Line 20 indicates that the TD error is also computed and stored. The target network is periodically replaced by the policy network after a certain number of learning steps (line 21). Line 22 updates the current state. Line 23 credits the reward to local profit. Line 24 updates the environment. If the local profit improves the global maximum profit (line 25), the global maximum profit is updated in line 26. Meanwhile, the paths of mobile users are recorded in line 27, and variable k is set to 0 in line 28. Otherwise, variable k is added by 1 (lines 29-30). Finally, line 31 returns the final result.

B. Baseline Solutions

For performance comparison, we propose three baseline solutions to the task allocation problem, *i.e.*, ant colony system (ACS), ϵ-greedy, and random solutions.

Ant colony system is a swarm intelligence method that has been successfully applied to many combinatorial optimization problems, *e.g.*, the travelling salesman problem [45,46]. ACS is a bio-inspired algorithm that imitates the behavior of real ants when they search for food. Each ant follows a path to its destination and releases a chemical, called pheromone, along the path. The pheromone, a communication medium, is used to guide other ants to find their paths. Since the pheromone evaporates over time, the path to a food source is frequently associated with the high-level pheromone and the ants are more likely to follow the path. Finally, the positive feedback of the pheromone leads to the optimal path for the ants.

Alg. 2 shows ACS solution to the task allocation problem. Lines 1-3 initialize the global maximum profit, number of episodes without improvement, and pheromone. Line 4 indicates that the searching process ends when there is no improvement found within M episodes. In each episode, a local profit is initialized in line 5. For each ant (line 6), there is an associated profit (line 7) and it sequentially selects actions (line 8) if there exist feasible ones (line 9). Here, the sensing tasks are ordered based on a rule of "first-arrive-first-assign". Then, the action selection follows the ϵ-greedy strategy in line 10. The profit of the selected assignment is added to the profit of the ant (lines 11-12). Line 13 updates the environment. The pheromone is locally updated for each ant in line 14. Lines 15-17 record the best result in the current episode. Then, the pheromone is globally updated for each episode in line 18. There are two types of updates for the pheromone, *i.e.*, local and global updates. More details concerned with updating the pheromone can be found in [45]. Lines 19-24 record the best result in all episodes and update variable k. Line 25 returns the final result.

In existing studies, greedy-based solutions are widely used [47,48]. Hence, we consider ϵ-greedy solution as a baseline, where the platform randomly selects mobile users with probability ϵ; Otherwise, the mobile user with the largest profit is selected. Alg. 3 shows details of ϵ-greedy solution.

In addition, we propose random solution as another baseline, which changes the ϵ-greedy selection (see line 7 in Alg. 3) to a pure random selection.

V. Numerical Results and Discussion

In this section, we evaluate the performance of DDQN-PER solution and the baseline solutions via simulations.

A. Simulation Settings

We set up the sensing region as a square area with side length of 100 meters. Then, 50 sensing tasks and 15 mobile users are uniformly distributed in this area. There are three different scenarios with 5, 10, and 15 mobile users respectively, shown in Fig. 3. Euclidean distance is used. The travelling speed of each mobile user is 1 meter per second (*i.e.*, normal walking speed of people) and the reward per meter (*i.e.*, λ_j in (1)) is 0.1 unit currency. Mobile users randomly appear in the sensing area between 0 and 30 (time unit is second). Their participation time lasts for 90 seconds. For each sensing task, the start time ranges from 0 to 60 (in seconds) and the length of the time window is 60 seconds. The budget of each sensing task is 3 unit currency.

The simulations end when there are 1000 episodes without improvement. In DDQN-PER solution (see Alg. 1), the capacity of replay memory is 10000. The probability of random selection starts at 0.9 and then gradually decays to 0.05 with a decay factor of 200. The target network is replaced by the policy network every 100 learning steps. The discount factor is set to 0.9. Both the policy network and the target network are DNNs with one hidden layer. Here, rectified linear unit (ReLU) is used in the DNNs. The dimensions of input layer and output layer depend on the numbers of sensing tasks and mobile users, while the size of the hidden layer is set to 128. The number of sampled transitions in a minibatch is also 128. In ACS solution (see Alg. 2), the number of ants is 10. The probabilities of random selection in both ACS solution and ϵ-greedy solution (see Alg. 3) are 0.05. Lastly, the settings of all simulation parameters can be found in Table II.

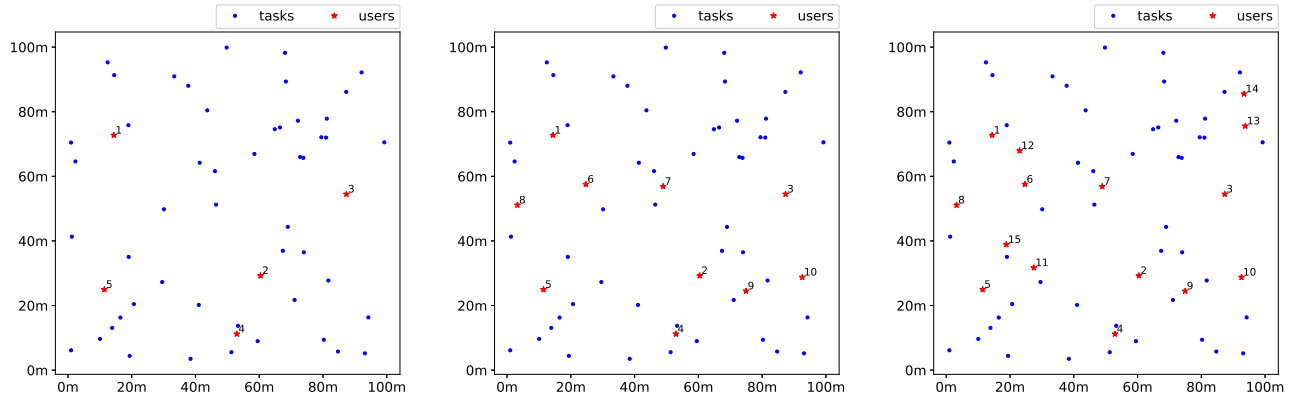
TABLE II
Simulation parameters.

Parameter	Value
Sensing area	100×100
Number of sensing tasks	50
Number of mobile users	5, 10, 15
Starts of tasks' time windows	0-60
Duration of tasks' time windows	60
Budgets of tasks	3
Starts of users' participation time	0-30
Duration of users' participation time	90
Travelling speeds of users	1
Rewards per distance unit of users	0.1
Number of episodes without improvement	1000
Capacity of replay memory in Alg. 1	10000
Start probability of random selection in (9)	0.9
End probability of random selection in (9)	0.05
Decay factor in (9)	200
Steps to update target network in Alg. 1	100
Discount factor in Alg. 1	0.9
Size of hidden layer in Alg. 1	128
Size of minibatch in Alg. 1	128
Number of ants in Alg. 2	10
Probability of random selection in Alg. 2	0.05
Probability of random selection in Alg. 3	0.05

B. Platform's Profit

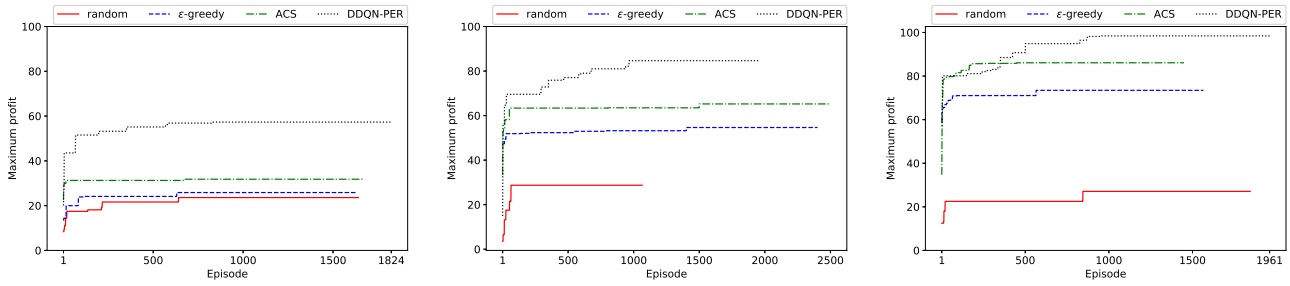
The platform's profit is the objective of the task allocation problem in (3). Thus, it is the most important indicator of the performance. Fig. 4 shows the maximum platform's profits over episodes, *i.e.*, the largest profit found before the current episode. We can see that DDQN-PER solution finally achieves the largest profit in all three scenarios. DDQN-PER solution significantly improves the maximum platform's profit at the beginning hundreds of episodes and then gradually converges to the final results. We can also clearly see when DDQN-PER solution outperforms the baseline solutions from the curves in Fig. 4, and the improving curves of DDQN-PER solution demonstrate its ability to solve the task allocation problem.

In order to compare different solutions more easily, Fig. 5 shows the final results in terms of the platform's profit. DDQN-PER solution attains the largest profits in all three scenarios, *i.e.*, 57.31, 84.66, and 98.44, respectively. Although ACS solution can leverage the historical information (*i.e.*, pheromone) in order to perform better than ϵ-greedy and random solutions, there is still a gap between DDQN-PER and ACS solutions. It is obvious that the increasing number of mobile users helps to improve the final platform's profits. This is because the platform can have more and better choices to assign sensing tasks if the additional mobile users participate in DeepSensing. However, the number of mobile users has little influence on random solution and the profits in three scenarios are 23.60, 28.73, and 27.10, respectively. We also observe that the gap between DDQN-PER and ACS solutions becomes smaller with the increasing number of mobile users, *i.e.*, 25.51 (5 users),



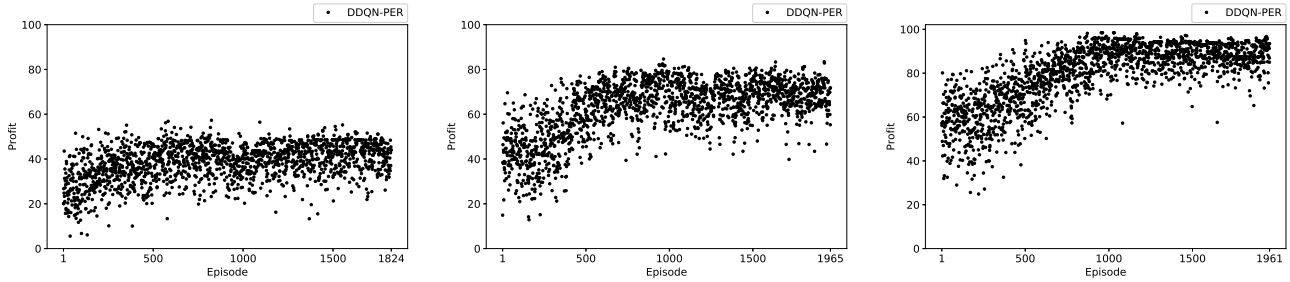
(a) 5 users. (b) 10 users. (c) 15 users.

Fig. 3. Simulation scenarios.



(a) 5 users. (b) 10 users. (c) 15 users.

Fig. 4. Maximum platform's profits over episodes.



(a) 5 users. (b) 10 users. (c) 15 users.

Fig. 6. Training processes of DDQN-PER solution.

19.42 (10 users), and 12.35 (15 users), respectively. The reason is that the redundant mobile users can reduce the difficulty of finding a satisfactory solution. The platform can easily find the appropriate mobile user for sensing tasks at a very low cost. Therefore, ACS solution with a weaker searching ability is approaching DDQN-PER solution in performance, due to the increasing number of mobile users.

Fig. 6 shows the training processes of DDQN-PER solution in the three scenarios. Each training process shows an increasing trend of the platform's profit over the number episodes. Although the learning process presents a general growing trend, many points with small profits during the process still show the exploration executed by DDQN-PER solution. In addition, the training processes in the three scenarios end at similar numbers of episodes, *i.e.*, 1824, 1965, 1961, respectively. This shows that the increasing number of mobile users in the simulations does not have a strong influence on the number of episodes required for the training.

C. Completed Tasks

A sensing task is considered to be completed if it is assigned and performed. The number of completed tasks is the important part of the performance in DeepSensing. On the one hand, it is related to the platform's profit because a larger profit can be obtained if more sensing tasks are completed. On the other hand, it prevents a loss of customers when data requesters drop out of DeepSesning because their published sensing tasks are not finished in the previous sensing activities. Fig. 7 shows the numbers of completed tasks in three scenarios. We observe that the number of completed

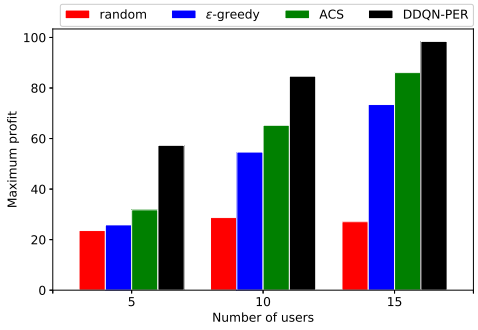


Fig. 5. Maximum platform's profits in all episodes.

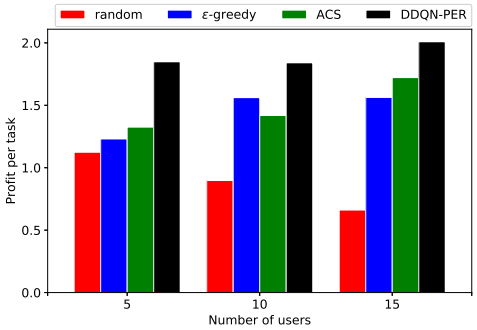


Fig. 8. Profits of each completed task.

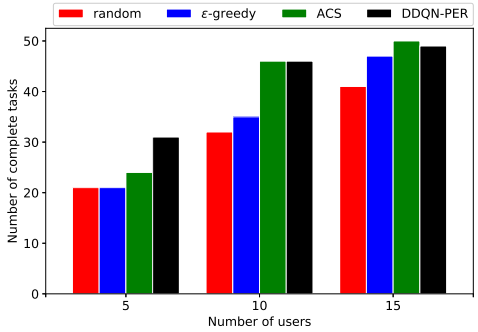


Fig. 7. Numbers of completed tasks.

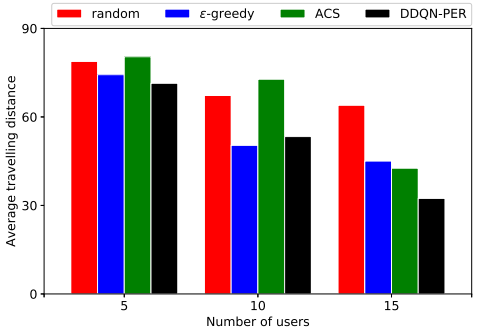


Fig. 9. Average travelling distances of users.

tasks increases with the number of mobile users. For instance, there are 31, 46, and 49 sensing tasks performed by 5, 10, and 14 mobile users in DDQN-PER solution, respectively. In the simulation scenario with 15 mobile users, both DDQN-PER and ACS solutions accomplish almost all sensing tasks (*i.e.*, 50 and 49). Thus, 15 mobile users are sufficient to perform the published sensing tasks in the simulations.

Although the numbers of completed tasks in DDQN-PER and ACS solutions are almost the same in the scenarios with 10 and 15 mobile users (see Fig. 7), the platform's profits in the two solutions are quite far apart (see Fig. 5). That is to say, the platform receives different profits from each completed task in these solutions. Fig. 8 shows the profits of each completed task in all three scenarios. We can see that DDQN-PER solution achieves the largest profits (around 2) in the three scenarios. Fig. 5, Fig. 7, and Fig. 8 indicate that DDQN-PER solution not only achieves the largest overall profit, but also obtains the largest average profit of each completed task.

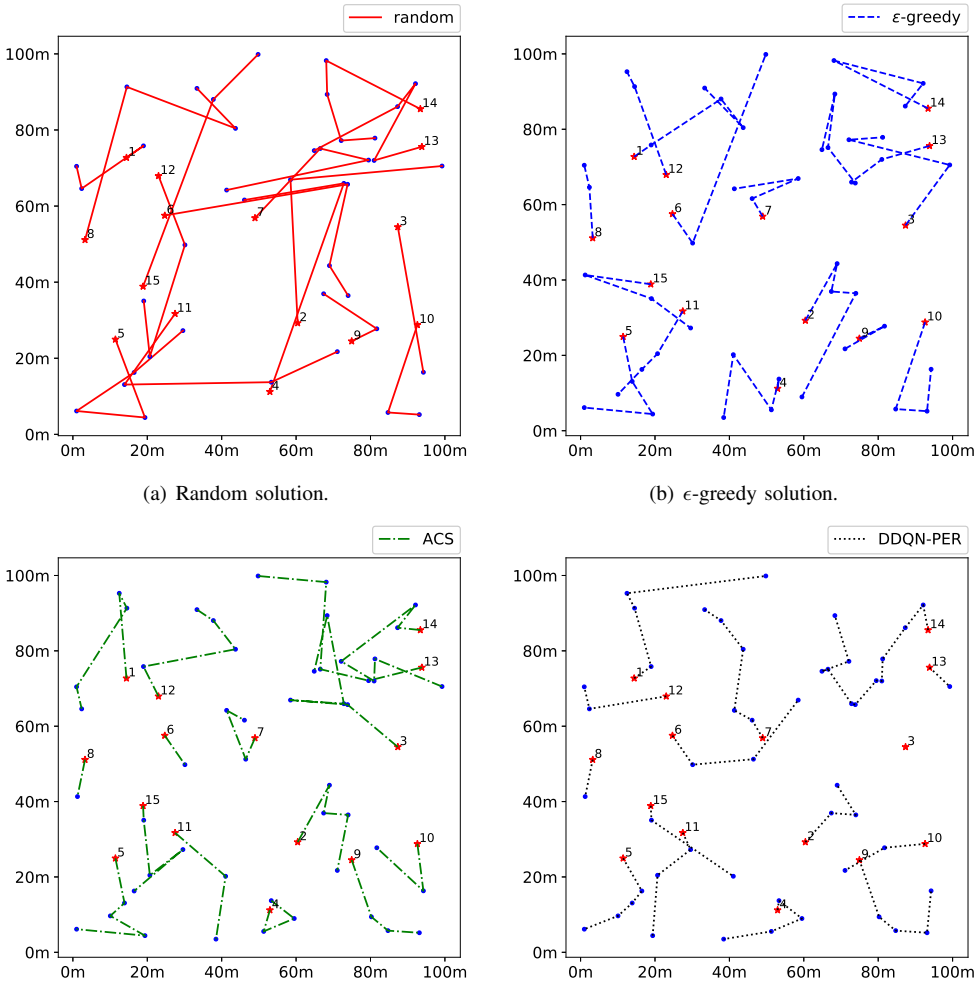
D. Travelling Paths of Users

Since the task allocation problem is also considered as a path planning problem, the planned paths of mobile users are critical for the platform's performance. Fig. 9 shows the average travelling distances of mobile users in three scenarios. We observe that the average travelling distances decrease with the numbers of mobile users. For instance, the average travelling distances of DDQN-PER solution in the three scenarios are 71.39, 53.34, and 32.37, respectively. There are less sensing tasks on average assigned to each mobile user when more mobile users are involved, and then mobile users can plan shorter paths to perform the assigned tasks. Fig. 9 also presents the fact that DDQN-PER solution has the smallest average travelling distance of mobile users in the three scenarios. As mentioned in (1), the rewards to mobile users are proportional to the travelling distances of their paths. That is to say, DDQN-PER solution distributes the smallest total reward to mobile users. Since the platform's profit is the difference between the total budget of sensing tasks and the total reward to mobile users, DDQN-PER solution achieves the largest profits (see Fig. 5).

To illustrate more details of the planned travelling paths, Fig. 10 shows the paths of mobile users in the scenario with 15 mobile users. We can see that the paths of mobile users are planned differently by the proposed solutions. The paths in random solution are the least organized, full of path-crossings due to its pure random nature. Compared with the paths in random solution, the paths in ϵ-greedy and ACS solutions are better-planned. However, there are still some path-crossings, *e.g.*, paths of mobile user #3, #13, and #14 in ACS solution. From the perspective of space, path-crossing means that some mobile users have to travel a long way to perform the assigned sensing tasks. This increases the cost and reduces the platform's profit. Finally, the paths in DDQN-PER solution are well organized almost without path-crossings.

VI. Conclusion

In this paper, we built DeepSensing as a novel MCS framework to efficiently assign sensing tasks to mobile users. We presented the structure of DeepSensing and formulated the task allocation problem in DeepSensing as a path planning problem. We also proved the formulated problem to be NP-hard. To optimally solve the task allocation problem,



(a) Random solution.

(b) ϵ-greedy solution.

(c) ACS solution.

(d) DDQN-PER solution.

Fig. 10. Travelling paths of users.

we proposed a solution based on double deep Q-network with prioritized experience replay from the perspective of decision making. We also presented three baseline solutions, *i.e.*, ant colony system, ϵ-greedy, and random solutions, for performance comparison. Finally, the results of numerical simulations show that our proposed solution outperforms the baseline solutions in terms of the platform's profit, and also plans better-organized travelling paths for mobile users.

REFERENCES

[1] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, 2011.

[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 Computing Surveys*, vol. 48, no. 1, pp. 7:1–7:31, 2015.

[3] Z. Pan, H. Yu, C. Miao, and C. Leung, "Crowdsensing air quality with camera-enabled mobile devices," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 4728–4733.

[4] X. Wang, Z. Ning, X. Hu, E. C. H. Ngai, L. Wang, B. Hu, and R. Y. Kwok, "A city-wide real-time traffic management system: Enabling crowdsensing in social Internet of vehicles," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 19–25, 2018.

[5] S. Jovanovic, M. Jovanovic, T. Skoric, S. Jokic, B. Milovanovic, K. Katzis, and D. Bajic, "A mobile crowd sensing application for hypertensive patients," *Sensors*, vol. 19, no. 2, p. 400, 2019.

[6] F. Restuccia, N. Ghosh, S. Bhattacharjee, S. K. Das, and T. Melodia, "Quality of information in mobile crowdsensing: Survey and research challenges," *ACM Transactions on Sensor Networks*, vol. 13, no. 4, pp. 34:1–34:43, 2017.

[7] H. Gao, C. H. Liu, W. Wang, J. R. Zhao, Z. Song, X. Su, J. Crowcroft, and K. K. Leung, "A survey of incentive mechanisms for participatory sensing," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 918–943, 2015.

[8] L. G. Jaimes, I. J. Vergara-Laurens, and A. Raij, "A survey of incentive techniques for mobile crowd sensing," *IEEE Internet of Things Journal*, vol. 2, no. 5, pp. 370–380, 2015.

[9] L. Kazemi and C. Shahabi, "Geocrowd: Enabling query answering with spatial crowdsourcing," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, 2012, pp. 189–198.

[10] X. Tao and W. Song, "Location-dependent task allocation for mobile crowdsensing with clustering effect," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 1029–1045, 2019.

[11] S. R. B. Gummidi, X. Xie, and T. B. Pedersen, "A survey of spatial crowdsourcing," *ACM Transactions on Database Systems*, vol. 44, no. 2, pp. 8:1–8:46, 2019.

[12] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015, pp. 21:1–21:10.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2011.

[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. 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, 2015.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] W. Guo, W. Zhu, Z. Yu, J. Wang, and B. Guo, "A survey of task allocation: Contrastive perspectives from wireless sensor networks and mobile crowdsensing," *IEEE Access*, vol. 7, pp. 78 406–78 420, 2019.

[18] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong, "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3747–3757, 2018.

[19] B. Guo, Y. Liu, L. Wang, V. O. K. Li, J. C. K. Lam, and Z. Yu, "Task allocation in spatial crowdsourcing: Current state and future directions," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1749–1764, 2018.

[20] D. Zhang, H. Xiong, L. Wang, and G. Chen, "CrowdRecruiter: Selecting participants for piggyback crowdsensing under probabilistic coverage constraint," in *Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2014, pp. 703–714.

[21] H. Xiong, D. Zhang, G. Chen, L. Wang, V. Gauthier, and L. E. Barnes, "iCrowd: Near-optimal task allocation for piggyback crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 2010–2022, 2016.

[22] W. Liu, Y. Yang, E. Wang, and J. Wu, "Dynamic User Recruitment with Truthful Pricing for Mobile CrowdSensing," in *Proceedings of the 39th IEEE International Conference on Computer Communications*, 2020.

[23] S. He, D. Shin, J. Zhang, and J. Chen, "Near-optimal allocation algorithms for location-dependent tasks in crowdsensing," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 4, pp. 3392–3405, 2017.

[24] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha, "Piggyback CrowdSensing (PCS): Energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, 2013, pp. 7:1–7:14.

[25] H. Xiong, D. Zhang, L. Wang, and H. Chaouchi, "EMC^3: Energy-efficient data transfer in mobile crowdsensing under full coverage constraint," *IEEE Transactions on Mobile Computing*, vol. 14, no. 7, pp. 1355–1368, 2015.

[26] H. Xiong, D. Zhang, L. Wang, J. P. Gibson, and J. Zhu, "EEMC: Enabling energy-efficient mobile crowdsensing with anonymous participants," *ACM Transactions on Intelligent Systems and Technology*, vol. 6, no. 3, pp. 39:1–39:26, 2015.

[27] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics*, vol. 59, no. 8, pp. 1207–1223, 2006.

[28] D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[29] L. Wang, D. Zhang, A. Pathak, C. Chen, H. Xiong, D. Yang, and Y. Wang, "CCS-TA: Quality-guaranteed online task allocation in compressive crowdsensing," in *Proceedings of ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 683–694.

[30] Z. Liu, Z. Li, and K. Wu, "UniTask: A unified task assignment design for mobile crowdsourcing-based urban sensing," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6629–6641, 2019.

[31] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, "ActiveCrowd: A framework for optimized multitask allocation in mobile crowdsensing systems," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 3, pp. 392–403, 2017.

[32] E. Wang, Y. Yang, J. Wu, W. Liu, and X. Wang, "An efficient prediction-based user recruitment for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 16–28, 2018.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[34] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[35] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274

[36] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[37] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[38] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[39] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Vehicular Technology Magazine*, vol. 14, no. 2, pp. 44–52, 2019.

[40] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for internet of vehicles," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 12–18, 2019.

[41] Y. Dai, D. Xu, S. Maharjan, Z. Chen, Q. He, and Y. Zhang, "Blockchain and deep reinforcement learning empowered intelligent 5G beyond," *IEEE Network*, vol. 33, no. 3, pp. 10–17, 2019.

[42] X. Tao and W. Song, "Task allocation for mobile crowdsesning with deep reinforcement learning," in *Proceedings of IEEE Wireless Communications and Networking Conference*, 2020, to appear.

[43] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics*, vol. 34, no. 3, pp. 307–318, 1987.

[44] M. G. Kantor and M. B. Rosenwein, "The orienteering problem with time windows," *Journal of the Operational Research Society*, vol. 43, no. 6, pp. 629–635, 1992.

[45] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[46] M. Dorigo and M. Birattari, *Ant Colony Optimization*. Springer, 2010.

[47] M. H. Cheung, R. Southwell, F. Hou, and J. Huang, "Distributed time-sensitive task selection in mobile crowdsensing," in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2015, pp. 157–166.

[48] X. Wang, W. Wu, and D. Qi, "Mobility-aware participant recruitment for vehicle-based mobile crowdsensing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4415–4426, 2018.

Xi Tao received both his B.Eng. degree and M.Eng degree in Electrical Engineering from Xi'an Jiaotong University, Xi'an, China, in 2013 and 2016, respectively. He received his Ph.D. degree in Computer Science from University of New Brunswick, Fredericton, NB, Canada, in 2020. He is currently a postdoctoral researcher in the Department of Computer Science and Operations Research, University of Montreal, Montreal, QC, Canada. His research interests include mobile crowdsensing, edge computing, and IoT.

Abdelhakim Senhaji Hafid is Full Professor at the University of Montreal. He is the founding director of Network Research Lab and Montreal Blockchain Lab. He is research fellow at CIRRELT, Montreal, Canada. Prior to joining University of Montreal, he spent several years, as senior research scientist, at Bell Communications Research (Bellcore), NJ, US, working in the context of major research projects on the management of next generation networks. Dr. Hafid was also Assistant Professor at Western University (WU), Canada, Research Director of Advance Communication Engineering Center (venture established by WU, Bell Canada and Bay Networks), Canada, researcher at CRIM, Canada, visiting scientist at GMD-Fokus, Germany and visiting professor at University of Evry, France. Dr. Hafid has extensive academic and industrial research experience in the area of the management and design of next generation networks. His current research interests include IoT, Fog/edge computing, Blockchain, and Intelligent Transport Systems.