Participants Selection for From-Scratch Mobile Crowdsensing via Reinforcement Learning

Yunfan Hu[1], Jiangtao Wang[1]*, Bo Wu[2], Sumi Helal[1]
[1]*School of Computing and Communications*, Lancaster University, UK
[2]*Columbia University*, USA
freddy@pku.edu.cn, {jiangtao.wang, s.helal}@lancaster.ac.uk, bo.wu@columbia.edu
* The corresponding author

***Abstract*—Participant selection is a major research challenge in Mobile Crowdsensing (MCS). Previous approaches commonly assume that adequately long and fixed periods of candidate participants' historical mobility trajectories are available before the selection process. This enables the frameworks to accurately model mobility which enables the optimization of selection. However, this assumption may not be realistic for newly-released MCS applications or platforms because the candidates have just boarded without previous mobility profiles. The sparsity or even absence of mobility traces will incur inaccurate location prediction of the individual participant, thus imposing negative effects on the participant selection process and hindering the practical deployment of new MCS applications. To this end, this paper investigates a novel problem called "From-Scratch MCS" (FS-MCS for short), in which we study how to intelligently select participants to minimize such "cold-start" effect. Specifically, we propose a novel framework based on reinforcement learning, which we name RL-Recruiter. With the gradual accumulation of mobility trajectories over time, RL-Recruiter can make a good sequence of participant selection decisions for each sensing slot by incrementally extracting and utilizing the collective mobility patterns of all candidate participants, thus avoiding the prediction of individual participant's location that is very inaccurate when the training data is sparse. We test our approach experimentally based on two real-world mobility datasets. Our experiment results demonstrate that RL-Recruiter outperforms the baseline approaches under various settings.**

***Index Terms*—mobile crowdsensing, participant selection, reinforcement learning**

I. Introduction

Mobile Crowdsensing(MCS) [1] [2] has emerged as a promising way of ubiquitous sensing and computing in recent years. In an MCS application or system, dynamically-moving users (called *participants or workers*) contribute location-dependent urban sensing information through various types of sensors embedded in their mobile devices. By collecting and aggregating each piece of information, those who publish and manage the sensing task (called *organizers* or *requesters*) can get an overall sensing picture within a certain spatial-temporal scale. Data quality and sensing cost are two main opposing concerns in MCS [3] [8] [29]. To get high data quality (e.g., spatial-temporal coverage), a naive way is to recruit as many participants as possible. However, it would incur excessive sensing costs (e.g., energy consumption and communication cost) [10], which translates to a high budget requirement where the organizers have to pay very high incentive rewards to compensate and attract participants. Therefore, optimized participant selection becomes one of the most crucial research problems in MCS, and many studies such as [4] [5] [6] [7] [8] [9] [11] [12] [15] [17] [18] keep focusing on this topic in recent years. They either aimed at minimizing the sensing cost with a fixed sensing quality requirement or attempted to maximize the sensing quality under sensing cost constraint. There are several metrics to characterize the sensing cost and quality [29], but to simplify the problem formulation, state-of-the-art research such as [5] [8] [9] [14] generally regards the sensing cost as being proportional to the number of selected participants and measures the sensing quality with spatial-temporal coverage.

With different optimization goals and constraints, most of these state-of-the-art research follows a common workflow consisting of two major steps. First, the candidate participants' historical mobility trajectories are used as training data to perform mobility pattern analysis so that their future locations can be predicted. Second, as the selection problem is NP-hard in nature, these approaches adopt various types of combination optimization algorithms (e.g., greedy search, dynamic programming, max flow, etc.) to select a near-optimal subset of participants (for single-task scenarios) [4] [8] or task-and-participant pairs (for multi-task scenarios) [10] [14].

Although these works differ from each other in either the mobility prediction methods or combination optimization algorithms, they hold a common assumption, that is, **there exists a fixed and adequately long period of candidate participants' historical mobility trajectories (usually more than a one-week period) as the input to facilitate the entire workflow** [3]. However, this assumption may be broken for newly-released MCS applications or platforms because the candidates are just onboard without previous working records and historical mobility traces. Regarding this issue, some research works simply assume that the new applications can obtain traces such as connected cell towers when calling/sensing text message, taxi mobility trajectories, check-in locations from other platforms (e.g. telecom operator, mobile social apps, taxi management systems, etc.) [8] [9] [18]. Unfortunately, this assumption sometimes tends to be impractical in real-world scenarios due to the following reasons. First, the candidate participants are often not willing to expose their historical mobility profiles in other mobile platforms to the MCS applications, as this is likely to incur the location privacy leakage [21]. Second, even if the candidates are willing to do so (e.g., motivated by attractive incentives), the MCS applications usually do not have data access to these platforms because these platforms do not have obligations to provide open location data collection APIs and may also concern about their own security risk [22]. Without the availability of those external mobility data, a newly-released MCS application or platform may inevitably face the challenge that, with sparse historical mobility trajectories, how to gradually learn the mobility patterns and intelligently select participants over time, which is referred to as *"From-the-Scratch MCS"* (FS-MCS for short) in this paper. A straightforward solution might be to let new participants move for, say, 1 week without paying them, or paying some fixed small amount, and then start paying normally when there is enough historical trajectory data. Although this solution might be feasible in some cases, it will lead to the delay of the sensing task execution, so that making it inappropriate for urgent MCS tasks.

In fact, for the participant selection in the first sensing slot, the random selection method is the best if we do not know any historical mobility trajectories or other relevant prior knowledge. With the accumulation of mobility trajectories over time, one straightforward method is to update the database of historical mobility trajectories whenever we get new mobility records in each sensing slot, and then directly adopt the state-of-the-art approaches (e.g., the framework named CrowdRecruiter [8]) based on the updated trajectory database to perform the participant selection for the next slot. However, this approach has a significant shortcoming before it gets enough training data. When the historical mobility trajectories are insufficient, the mobility prediction of each participant will be very inaccurate. Then, the inaccuracy of individual-level mobility prediction will have negative influences on the optimization of the whole participant selection process, which leads to relatively poor performance. Although no algorithms can eliminate the cold-start effect as no other prior knowledge or information is introduced, we attempt to answer the following research question: *can we try to minimize the cold-start effect for FS-MCS?*

To address this issue, we have the following intuitions. Though individual mobility is less predictable when the training data is sparse, the mobility of all crowd participants still possesses some implicit patterns. For example, people usually share some common commuting patterns (e.g., from residential areas to CBD). Furthermore, people tend to occur in the same locations due to their social ties (e.g., friends, colleagues, family members, etc). [32]. In other words, despite the mobility uncertainty at the individual level without enough training data, there exist some useful collective patterns. Thus, our basic idea is that, instead of predicting each participant's location, we leverage the latent collective mobility patterns of all candidates in the participant selection process. In other words, although we cannot eliminate the impact of cold-start effects as nothing new information is introduced, we seek an alternative to avoid the individual-level mobility prediction to improve the selection performance.

The above intuition is easy to understand, but the collective pattern of all candidate participants is implicit and subtle, so it is not straightforward to model and utilize it in our focused problem. To address this challenge, this paper exploits the idea of reinforcement learning in the AI area and proposes a novel framework, called RL-Recruiter. Specifically, RL-Recruiter first models the key concepts in reinforcement learning (i.e., state, action, and reward) according to our formulated problem. Then, based on this model, RL-Recruiter selects a pre-defined number of participants in each sensing slot through a reinforced self-learning way, which consists of three main steps. First, in a certain sensing slot, RL-Recruiter explores and exploits different actions of selection and trains the decision model with returned reward based on the achieved spatial-temporal coverage in previous sensing slots. Second, with the designed updated decision model, RL-Recruiter iteratively selects participants in the current sensing slot. In each iteration, it takes an action (i.e., adding one participant to the selected set) with the maximum action value. Third, at the end of the sensing slot, RL-Recruiter observes the sensing task execution outcome (i.e., the real trajectories and the finally achieved spatial-temporal coverage within this slot). This observation is used to improve the knowledge of RL-Recruiter, that is, updating the dataset to train the decision model for future sensing slots. Compared to existing approaches, the advantage of RL-Recruiter is that it can remove the strict assumption about the pre-existence of adequately long and fixed periods of mobility profiles, thus improving the practical value of MCS in real-world pervasive sensing and computing scenarios.

To illustrate our formulated problem and how RL-Recruiter works, we present a running example as follows: *The city government launches an MCS application, called AirSense, for collecting real-time air quality information in different regions every two hours as a sensing slot in the city with a budget constraint (e.g., 2000 GBP per sensing slot). The entire sensing area can be divided into 20 subareas, and a total of 500 mobile users living in or near these areas have registered as candidate participants in AirSense. Selected users will use mobile phone embedded with air quality sensors [38] or connected to portable sensing box [36] [37] to collect air quality data and upload the data with the application installed. As AirSense is just newly released, the application knows nothing about the participants' historical mobility traces at the very beginning, but their mobility traces will be gradually recorded over time and utilized by AirSense only for the purpose of participant selection after proper anonymization. As the budget is limited and assuming that each participant is equally paid (e.g., 10 GBP per participant), the goal of AirSense is to select a pre-defined number of participants (2000/10=200 participants) in each two-hour sensing slot to maximize the average spatial-temporal coverage over time. To address this problem, AirSense adopts the RL-Recruiter framework for participant selection, which makes a good sequence of selection decisions for each sensing slot. For the first sensing slot, as no information is known, RL-Recruiter* *will randomly select 200 participants. From the second sensing slot, before the task execution of a given slot, RL-Recruiter will explore different selection possibilities and determine a selected participant set with experience. At end of the slot, it will observe the actual outcome (i.e., real mobility trace of the participants and the achieved spatial-temporal coverage within this slot) and update its knowledge towards the direction that benefits the selection of coming slots. The above process will repeat for future slots (3rd, 4th, ...).*

In summary, our work makes the following contributions. First, through the analysis of existing frameworks of MCS participant selection, we argue that the common assumption about the pre-existence of mobility profiles may not be realistic in cold-start situations. Then, we formulate a novel problem called "From-Scratch MCS" (FS-MCS). FS-MCS aims to study how to minimize the cold-start effect of mobility prediction on the participant selection problem in MCS. That is, how to make a good sequence of participant selection decisions with the gradual accumulation of mobility trajectories over time.

Second, we propose RL-Recruiter, a novel framework based on reinforcement learning, to select a suitable set of participants in each of the sequential sensing slots. Before the task execution step in each given slot (from the second sensing slot), RL-Recruiter first explores and exploits different actions of selection and trains the decision model with returned reward based on the achieved spatial-temporal coverage in previous sensing slots. Then, it explores the possible actions of participant selection and uses the delayed outcome (i.e., the achieved spatial-temporal coverage within this slot) to improve its knowledge in future sensing slots.

Third, through extensive simulations utilizing two real-world mobility datasets, we demonstrate that RL-Recruiter outperforms the baseline approaches under various settings. Compared to the baseline methods, RL-Recruiter achieves a higher spatial-temporal coverage with the same number of participants in terms of both overall and slot-level comparison.

II. Related Works

Participant Selection for MCS. The state-of-the-art research of participant selection for MCS can be divided into two categories. (1) *Participatory* MCS. This category of sensing tasks requires the candidates to continuously report their real-time locations, and then the system selects the participants online. If a participant is selected and assigned to certain task locations, he/she will change their original routes and specifically move to the required places with certain incentive models [15] [16] [17] [18] [19]. (2) *Opportunistic MCS*. In this category, the participants do not need to report their locations and change their original trajectories. Instead, based on the modeling and prediction of the participants' mobility, the system selects the participants offline (i.e., before the execution of sensing tasks), and the participants simply complete the tasks during their daily routines. For example, the authors studied participants selection or recruitment for a single task, and they proposed different strategies to select a predefined number of participants so as to maximize the task's sensing quality [4] [5] [6] [7], or select a minimum number of participants to ensure a certain level of sensing quality [8] [11]. Other studies attempted to optimize the overall utility of multiple concurrent sensing tasks in a multi-task-oriented MCS platform where tasks share the limited resources [12] [13] [14] [20] [30]. Actually, RL-Recruiter falls into the opportunistic category. The above works of this category hold a common assumption, that is, relatively adequate historical moving trajectories of the candidate participants must be taken as the input to facilitate mobility prediction and participant selection. On the contrary, RL-Recruiter removes this assumption and investigates how to make a good sequence of participant selection decisions with the gradual accumulation of mobility trajectories over time.

Learning-Assisted Crowdsourcing/MCS. In recent years, there has been a new trend to optimize the crowdsourcing or MCS by integrating different types of machine learning techniques [23]. In these works, multiple aspects of MCS are self-learned from multiple trials over time. For example, in the focused scenario of [24], the authors assume that the participants decide whether to accept the task based on the incentive reward and movement distance. They developed a supervised learning method to model the relationship between task acceptance rate and these two factors, and then utilize it to design better incentive mechanisms, to reduce sensing cost while ensuring task completion. Han et al. [25] proposed an online learning approach to acquire statistical information about the sensing values from participants throughout the selection process. The authors in [26] presented an online algorithm that leverages the historical performing records of participants to learn the data upload time delay. The authors in [27] addressed the online labeling problem in which the ground truth is unknown, and they proposed an online algorithm using majority voting over time. Zhang et al. [28] considered expertise-aware task allocation and truth analysis in MCS where user expertise is estimated via an online learning framework. From the perspective of using learning techniques to assist crowdsourcing/MCS tasks, the above-mentioned studies share similar high-level ideas with our work. However, both our formulated problem and proposed RL-Recruiter framework are quite different from these works, which are complementary to the above studies.

Cold-Start Problem in Recommender Systems. In the recommender systems, a cold start happens when new users or new items arrive in e-commerce platforms. Classic recommender systems like collaborative filtering assumes that each user or item has some ratings so that we can infer ratings of similar users/items even if those ratings are unavailable. However, for new users/items, this becomes hard because we have no browse, click or purchase data for them. Therefore, researchers have proposed various ways, which can be roughly divided into the following categories [35]. (1) Representative-based-approach: use subset of items and users that represents the population; (2) Content-based approach: use side information such as text, social networks, etc. However, these approaches are not suitable for our from-scratch crowdsensing as we have

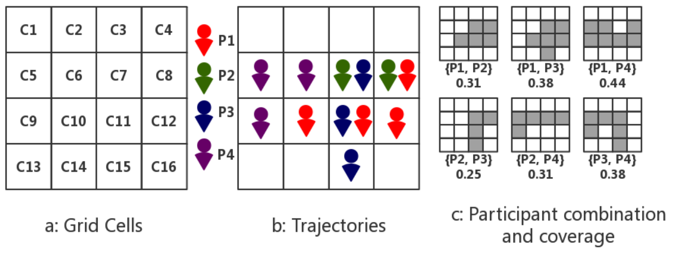


Fig. 1. An example of coverage condition in one time slot for four participants. The left grid illustrates the serial number of the cells, and the middle one shows the trajectories of participants in one time slot. The girds on the right display the covered cells given a specific participant combination. We can find that the combination $P = \{p_1, p_4\}$ covers seven cells and has the highest coverage $7/16 = 0.44$. Therefore, it is the optimal combination while the goal is to maximize the coverage for two participants.

neither representative participants nor the side information.

III. RL-Recruiter: Method Overview

A. Preliminary of Participant Selection

The participants will execute the sensing tasks in a sensing area which can be described as a rectangle geographical location. As the trajectories of participates associate with the attached times, we denote the time range of the trajectories into equal-length slots $(t_1, t_2, t_3, ...)$. When the participant p_k appears in the i-th cell c_i in the j-th slot t_j, the cell is reserved by the participant.

Given a target area with a set of grid cells T and the total set of available participants Z, we aim to select a set of participants P ($P \subseteq Z$) respectively in each slot t_j. If we represent the set of covered cells for participant p_k in t_j slot as $C_{(\{p_k\},t_j)}$, the covered cells set of all selected participants can then be denoted as $C_{P,t_j} = \bigcup\{C_{(\{p\},t_j)} | p \in P\}$. The goal of our model is to maximize its spatial-temporal coverage (”coverage” for short):

$$\begin{aligned} maximize \quad & |C_{(P,t_j)}|/|T| \\ s.t. \quad |P| = n \quad & j \in \{1,\ 2,\ 3,\ ...,\ m\} \end{aligned} \tag{1}$$

where n is the predefined number of participants to select, and m is the total number of slots.

In the Fig. 1, we show an example for participant selection. We split the sensing area into smaller grid cells $T = \{c_1, c_2, c_3, ..., c_{16}\}$ with equal size, as shown in Fig. 1(a). The Fig. 1(b) shows the trajectories of participants in the time slot t_j, and $Z = \{p_1, p_2, p_3, p_4\}$ is participant set. If we considers all of possible covered area, then the covered cells set C_{Z,t_j} is the upper range for possible covered cells in the slot t_j. The Fig. 1(c) shows trajectories of participant combinations.

B. Overview of RL-Recruiter

To solve the problem with the reinforcement learning strategy, we first need to introduce the three main concepts of our strategy: *state*, *action* and *reward*. Under the different states, an action may have different values determined by a decision model Q, that maps states and actions to values indicating the rewards after taking these actions. These concepts are shown in detail in section IV-A.

The framework for RL-Recruiter is shown in Fig. 2. It splits the process of the RL-Recruiter into three stages. The first two stages have a stop criterion, to cease the selection when the amount of selected participants reaches n. (i.e., the length of participant set $|P| = n$).

(1) **Decision Model Learning**. Before the start of the j-th slot ($j = 2, 3, ...$), RL-Recruiter explores and exploits different actions of selection and trains the decision model with returned reward. This reward is calculated based on the state and the achieved coverage in $(j-1)$-th slot. For each epoch of the training, we choose the participant and get the reward one after another to train the model until the stop criterion is met. The selected participants will not be assigned tasks.

(2) **Iterative Participant Selection**. In this stage, the updated decision model selects the participants iteratively to execute the tasks on the current slot. The decision model greedily takes the action with the highest value, until the stop criterion is satisfied and outputs the combination of participants.

(3) **Sensing Task Execution**. The selected participants execute their tasks in the current slot. Their daily routines cover some of the cells where participants finish their jobs. The coverage result of the tasks is obtained after this slot, and the trajectories of the participants are collected and used to learn the decision model again for future sensing slots.

IV. Core Components of RL-Recruiter

This section firstly explains the state, action, reward and decision model, then it introduces the details of the RL-Recruiter algorithm, especially for decision model learning and iterative participant selection.

A. Formulation of State, Action, Reward, and Decision Model

To illustrate the RL-Recruiter clearly, the three key concepts, *state*, *action* and *reward*, need to be defined in details:

State represents the participant selection condition of FS-MCS task. In RL-Recruiter, there are $|Z|$ participants in the target sensing area, and the state is a $|Z|$-dimension binary vector s. The initial values in the state vector are all zeros. When the k-th participant p_k is selected, the value in k-th dimension of the vector is set to one. The number of different state vector is $2^{|Z|}$.

Action means all the possible decisions that we may make in participant selection. The whole action set is $A = \{a_1, a_2, ..., a_n\}$, with each a_k attached to a value $Q(s, a_k)$ under a certain state s. If the k-th dimension of s is not set to one, then the action k, meaning selecting k-th participant, is available. $Q(s, a_k)$ suggests the probable reward after selecting the participant p_k under the state s.

Reward is used to indicate how good an action is. In each slot, RL-Recruiter takes actions one by one until selected participants satisfy the stop criterion for the current slot.

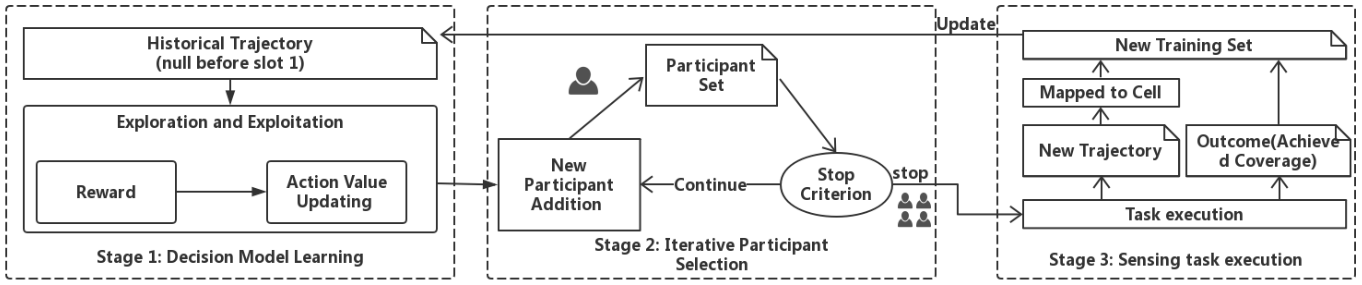

Update
Historical Trajectory (null before slot 1)
Exploration and Exploitation
Reward
Action Value Updating
Stage 1: Decision Model Learning
Participant Set
New Participant Addition
Continue
Stop Criterion
stop
Stage 2: Iterative Participant Selection
New Training Set
Mapped to Cell
New Trajectory
Outcome(Achieved Coverage)
Task execution
Stage 3: Sensing task execution

Fig. 2. Framework of the RL-Recruiter. Stage 1: the model learns the action values by selecting the participants and getting the rewards. Stage 2: the model iteratively selects participants with the highest action values. Stage 3: the selected participants execute the sensing tasks, and the coverage result and new trajectories are obtained to train the model again for future slots.

Maximizing coverage while meeting this criterion is the goal of the RL-Recruiter and should be reflected in the reward modeling. Hence, we set the reward, denoted by r, as the amount of increase in covered cells after one action is taken. Based on the reward and the highest action value that the model predicts to get for all available actions $Q(s^{'}, a)$ under the next state $s^{'}$, RL-Recruiter refreshes the value $Q(s, a)$ for each action it takes during the training process.

Decision model Decision model contains two components. One is a value function and the other is a policy. Value function Q maps current state and actions to values which indicate the gains of these actions. Under the current state vector s, the function outputs the values $Q(s, a)$ for all actions. The value $Q(s, a_k)$ is set to zero if a_k was taken before and the k-th dimension of the s is set to one, to avoid selecting the participant that has been already selected before. The optimal value of an action that lets the value function to get the highest performance is this:

$$Q(s, a) = r + \gamma maxQ(s^{'}, a^{'}) \tag{2}$$

where r is the reward, and $maxQ(s^{'}, a^{'})$ denotes the highest value of actions under the state of next turn $s^{'}$. γ is the discount rate of the future highest value which indicates the gain after taking the probably best action under the new state $s^{'}$. The other component, policy $\pi : s \to a$, makes the selection decision based on the value function. The way how policy utilizes the value function is changing when moving into different stages of the framework of the RL-Recruiter. In the decision model learning stage, the policy tends to try various combinations of participants, while in the iterative selection stage, the policy selects participants with the highest action value.

B. Decision Model Learning

In each slot t_j, the decision model goes through e epochs to perform the learning tasks. In each epoch, the decision model picks participants one after another until the stop criterion is met, and it updates value function based on the rewards that come from the process of selection. This process is set to let the model explore the datasets in the t_{j-1} slot, and capture the trajectory patterns of the participants.

We maintain a lookup table with l rows and $|A|$ columns to build the value function $Q(s, a)$. Under the current state s, the decision model needs to first calculate the corresponding index d of the row in the table, and the value in the k-th column of that row is just the value of the action a_k. d is determined by the equation:

$$d = \lfloor \frac{\sum s_k}{n} \times l \rfloor \tag{3}$$

where s_k is the value in k-th dimension of the state s, initialized as zero and set to one when k-th participant is selected. $\sum s_k$ is the number of the selected participants, and n is the number of total participants that we need to select eventually. The intuition of the equation is that we assume when the number of the selected participants changes, which is suggested by the fraction $\sum s_k/n$, the rewards for the rest of the actions change consequently. So, the index of the row is related to the fraction $\sum s_k/n$. This method is called *State Aggregation*, a kind of value-function approximation in reinforcement learning. We have to approximate the value because there are so many states in our problem, namely $2^{|Z|}$, as mentioned before. If we use tabular solutions, keeping values of all state-action pairs, then the space complexity will be $O(2^{|Z|} \times |Z|)$. This complexity is unmanageable. Instead, our approach gets the space complexity $O(l \times |Z|)$, while still maintaining the reinforcement learning framework, mapping the states and actions to the rewards. In this approach, l is a predefined number. The actual value of the l depends on the performance of the model in the runtime. The maximum value of the l is n, which means after one participant is selected, we need to change the row in the lookup table to check the new values for the next state.

The $\epsilon - greedy$ method is designed in the policy π to select participants during the training process. ϵ is a hyper-parameter within $(0, 1)$. It means the probability for the model to explore the data is ϵ, and that to exploit the data is $1 - \epsilon$. The model picks up the participants randomly in the exploration method and picks up the participants with the highest action value in exploitation methods. The reason to apply the $\epsilon - greedy$ method is that the model needs to explore the data and to find the maximum result. If the model always takes actions with the highest action value, then it will converge quickly while not fully understanding the data. Therefore, the result of it is less likely to be the global optimal one. The $\epsilon - greedy$ method helps the model better understand the patterns.

Algorithm 1: Decision Model Learning

```
participant set $P \leftarrow \emptyset$ ;
state vector $s$ ;
while $|P| < n$ do
    Calculate the index of the row in lookup table $d$;
    Get a random number $x$ between 0 and 1;
    if $x > \epsilon$ then
        Select the participant $p_k$ with highest action value;
    else
        Randomly choose one $p_k$ from all available participants;
    end
    Get the state of next step $s^{'} \leftarrow newState(p_k, s)$ ;
    Obtain the reward $r \leftarrow Reward(p_k, s)$ ;
    Update the model $Q(s, a_k) \leftarrow r + \gamma max Q(s^{'}, a^{'})$ ;
    Add $p_k$ into $P$ ;
    Update the state vector $s$ ;
end
```

After selecting the participant p_k, the model gets a reward r based on trajectory in t_{j-1} slot and the current state. This reward value is defined as:

$$r = |C_{(\{p_k\},t_j)} - C_{(P,t_j)}| \tag{4}$$

where P represents the set with all the selected participants previously. r is the number of increase in covered cells after the participant is added. This new selected participant will be added into P after the r is obtained.

The value of the taken action will be recalculated and this refreshed value will be used to predict the reward of this action in the future. Here the new value for action a_k is denoted by $Q(s, a_k)^{'}$, and it is calculated by:

$$Q(s, a_k)^{'} = Q(s, a_k) + \alpha(r + \gamma max Q(s^{'}, a^{'}) - Q(s, a_k)) \tag{5}$$

where the $Q(s, a_k)$ is the current value for action a_k under the state s, and the r comes from the equation 4. α is the learning rate to update the action value. The state vector of next turn $s^{'}$ is obtained by setting the value of the k-th dimension in the s to one, which means participant p_k is selected.

The model is trained with e epochs. In each epoch, the decision model may scan the whole participants to select one iteratively until there are n participants selected. Therefore, the time complexity of decision model learning is $O(e \times |Z| \times n)$. The pseudo-code of decision model learning in one epoch is shown in algorithm 1.

C. Iterative Participant Selection

As the decision model is updated, by using equation 5, the decision model selects a new participant set to do the FS-MCS task for current slot t_j. Iteratively, the policy π greedily picks up the participant with the highest action value. There are two differences between decision model learning and iterative participant selection. Firstly, iterative participant selection does not have the $\epsilon - greedy$ method. We assume that the model has already understood the trajectory pattern of the previous slot in the decision model learning process, so we take every action with the highest action value to gain the possibly highest final result. Secondly, the model has no rewards for all actions because there is no trajectory for current yet. The trajectory is only gained after the sensing tasks are finished at the end of the current slot when the rewards can be inferred by the newly gathered data. For iterative participant selection, the model picks the participants with the highest values one after another until there are n participants selected. Thus the time complexity is $O(n \times |Z|)$.

V. Evaluation

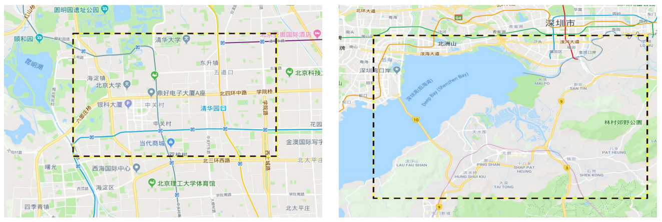
A. Metrics and Comparison Methods

In this experiment, we use relative coverage instead of absolute coverage to measure the performance of different methods. In each slot t_j, the relative coverage for participant set P is defined as $|C_{P,t_j}|/|C_{Z,t_j}|$, where $|C_{P,t_j}|$ represents the number of covered cells by the selected participant set P, and $|C_{Z,t_j}|$ represents the number of covered cells by all participants. Take the scenario in Fig. 1 as an example. If selected participant set $P = \{p_1, p_4\}$, then the relative coverage of the P is $|C_{P,t_j}|/|C_{Z,t_j}| = 7/9 = 0.78$, where the $Z = \{p_1, p_2, p_3, p_4\}$. The reason is that absolute coverage may be disturbed by a lot of factors (e.g., the choice of dataset, experimental settings such as sensing areas and time). The relative sensing coverage can better reflect performances of different models, and it is more independent from other factors.

In our evaluation, we apply two methods as baselines, and all of them share the same iteration process and stopping criterion, and they all use the data from previous slots to update the selection model at the beginning of the j-th slot. The first is CrowdRecuiter, one of the state-of-art methods the do the participant selection task in MCS. The second is DQN-Recruiter, the variant of our methods, which also uses reinforcement learning to train the decision model, except that it uses a neural network to build the decision model.

CrowdRecruiter. The original CrowdRecruiter [8] has two steps. Firstly it predicts the probability for every participant to cover each cell in the next slot respectively by referring to the Poisson intensity, based on the historical mobility trajectories. Secondly, the model iteratively adds the participant to the participant set to find the participant combination with the highest probability to cover the maximum cells. To apply this method in the FS-MCS problem, when we get new mobility records in each sensing slot, we update the database for training, and then directly adopt the CrowdRecruiter.

DQN-Recruiter. This method uses a neural network to do the reinforcement learning [31]. It is also an approximate solution designed to solve problems with large state space and imperfect information and it has delayed credit assignment requiring long-term strategies over thousands of steps. To



a: GeoLife b: Shenzhen Truck

Fig. 3. Target sensing area

apply it to the FS-MCS, we propose the DQN-Recruiter. It has a fully connected neural network just like the value function Q in the RL-Recruiter that maps states and actions to values. The input of the neural network is the same state vector s used by RL-Recruiter, and the outputs of the neural network are the values of all actions. Moreover, the neural network also uses a weighted sum of the reward and future value to update its output. The future value is the highest action value output by the neural network when it takes a state vector of the next turn as the input. When the model gets new rewards, it will update the neural network parameters to let its output fit the reward. The new state vector is updated from the current state just like the way in RL-Recruiter. Practically, we use tricks mentioned in [31] to train the network. For example, to avoid fluctuation in model parameters and break the dependency among the training data, we use two networks in the training process, one for decision making and the other for updating parameters. Also, we randomly pick up training data from the training set.

B. Datasets and Experiment Settings

In this section, we firstly introduce the two datasets to be used in evaluation, and then explain the setups we apply in the datasets. Besides, we give crucial parameter settings for the three models. Finally, we show our experiment environment settings, including hardware and software.

Datasets and Pre-processing. There are two data set used to evaluate models, *GeoLife GPS Trajectories*(named as GeoLife) [33] and *Truck GPS data of the Chinese City Shenzhen*(named as Shenzhen Truck). [34]

The GeoLife was collected in the (Microsoft Research Asia) Geolife project by 182 users in a period of over three years (from April 2007 to August 2012). A GPS trajectory of this dataset is represented by time-stamped points, each of which contains the information of latitude, longitude. This dataset contains 17,621 trajectories with a total distance of about 1.2 million kilometers and a total duration of 48,000+ hours.

The Shenzhen Truck contains 1000 participants. Each line in this data has 4 columns of information: Truck ID, Date-Time, Longitude, and latitude. It is a truck trajectory covering Shenzhen and the surrounding areas. This dataset is organized regularly. It contains about 10 weeks of trajectories and the observations in the same week are arranged together, all of which have column Truck ID, with values from 000 to 999. For each truck, its trajectories in the same week are listed together ordered by the date-time.

As the trajectories of two datasets cover various areas, we select one specific square area for each dataset respectively, to measure the participants' coverage and in this area and compare the performance of the different models. This is because the distribution of the participant trajectories is uneven, so we choose one area with the most concentrated trajectories to obtain denser covered cells of participants in this area. Additionally, a square area can simplify the problem because we can easily divide the square into equal-size cells.

For the first dataset GeoLife, we select the area in Haidian, Beijing, as is shown in Fig. 3(a). This square is in the northern latitude from 39.975 to 40.025, and eastern longitude from 116.31 to 116.35. We divide the square into 100×100 cells with equal size. As the date of the trajectories are not successive, we concatenate trajectories on discrete days for each participant, so long as the trajectories are in the same year, until there are trajectories in 10 days for one participant. Particularly, we select workdays and take out weekends because the pattern of the human movement in the weekdays differs from that in weekends. If a participant does not have enough days, we just copy the trajectories on the last day to be the new day's data until this participant has sufficient trajectories. This is because the data is sparse and the days of the trajectories are dispersing. We use trajectories from 01/01/2008 to 12/30/2011 and get 4 test periods. One test period has 10 days, and one day has 2 time slots. The first slot is from 0:00 to 8:00 and the second slot is from 8:00 to 20:00. These two slots have the relatively dense user trajectories. There are 20 slots in one test period. We use the average performance of 4 test periods to evaluate models.

The second dataset Shenzhen Truck is a larger one with 1000 participants, and the area we select is shown in Fig. 3(b). It is in the northern latitude from 22.2 to 22.7, and east longitude from 113.75 to 114.25. Some of the area that the dataset covers is in Hongkong, for these trucks always travel between Shenzhen and Hongkong. We divide the square into 100×100 cells with equal size. The square area is larger because the trucks can move to distant places, thus their trajectories cover a wider area than that of human. The trajectories in this dataset are combined weekly. Since the change in workdays to weekends has very little impact on the trajectory of trucks, we do not filter out the weekends. We set 10 days to be one test period and use the average performance of 5 test periods to evaluate models. In each day, there are 4 time slots, the first from 0:00 to 6:00, the second from 6:00 to 12:00, the third from 12:00 to 18:00, the forth from 18:00 to 24:00. Therefore, there are 10 days with 40 slots in one test period. The Shenzhen data have more time slots for its denser trajectories than GeoLife data.

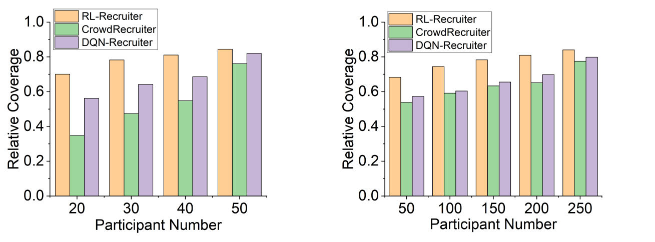
As mentioned above, RL-Recruiter and baseline models are trained based on data in the previous slot, and then select participants to do the tasks in the current slot. In the experiments, each dataset has its unique settings for models to use. For example, in GeoLife, there are two slots in each day. To ensure a better evaluation process, we train two sets of models to do the test, one for the first period, and the others for the second period. The DQN-Recruiter and RL-Recruiter are trained based on data in $(j-2)$-th slot ($j = 3, 4, 5, ...$), to capture pattern from the same period in a day, and then select participants to do the tasks in j-th slot. The CrowdRecruiter uses data from all slots in the same period in a day to do the tasks in j-th slot (from $j-2, j-4, j-6, ...$ slots). Similarly, we train four sets of models in Shenzhen Truck experiment because there are 4 slots in a day. The DQN-Recruiter and RL-Recruiter are trained based on data in $(j-4)$-th slot, and then select participants to do the tasks in j-th slot ($j = 5, 6, 7, 8, ...$). The CrowdRecruiter uses data from all slots in the same period in a day to do the tasks in j-th slot (from $j-4, j-8, ...$ slots).

Parameter Selection. For RL-Recruiter, we set the parameters in their values of best performance, which are obtained by altering one parameter with others fixed, to check the change in model performances. We test these parameters in two datasets under various settings. The ϵ in the $\epsilon - greedy$ method is set to be 0.8, which means the model tends to explore the data in the training process. And the epochs for decision model learning e is set to be 900, which is shown to be enough for the model to converge. The l in equation 3 is set to 100. The model can get higher performance when the l rises to approach the value of n, the number of participants we need to select. However, the rise of this parameter can result in a higher demand in time and space, while the gain in the performance is insignificant, and we find 100 is a proper value. The γ in equation 4 is set as 0.8 and the α in equation 5 is set as 0.1. There are no crucial parameters need to be illustrated in CrowdRecruiter. The neural network in DQN-Recruiter is a fully connected network with an input layer, 2 hidden layers, and an output layer. The number of nodes in the input and output layer are equal to $|Z|$, the total number of participants in datasets. Structure is set like this because the neural network acts like value function $Q(s, a)$ that map states and actions to values. When we put state vector s to the neural network, it outputs the value vector, whose value in the k dimension stands for the value of action a_k for the state s. The settings for hidden layers differ among various datasets and they are fine-tuned based on the performance of real datasets. We consider both model performance and time efficiency to determine the relevant parameters. The nodes in hidden layers are set to be 64 for GeoLife, and 256 for Shenzhen Truck.

Experiment Environment. To evaluate three methods, we use a single server to do all the running jobs. The CPU is Intel Xeon(R) E5-2560, and GPU is GeForce 1080Ti, with the 31G memory. All the coding is on the Python language. The neural network in DQN-Recruiter is built and trained using the open-source package Keras.

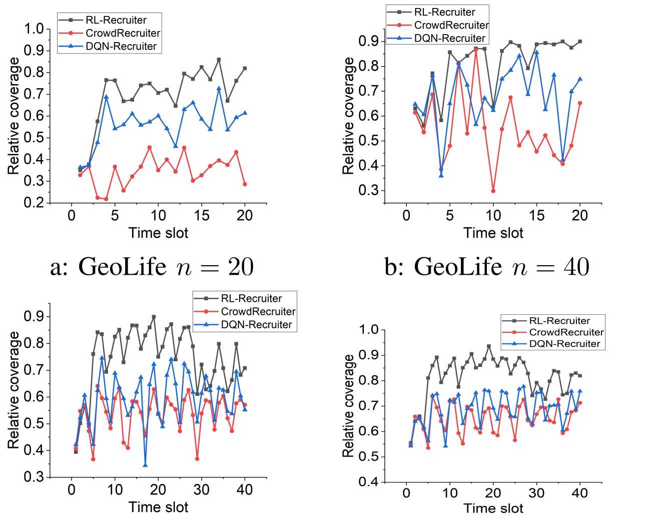
C. Experiment Results

In this section, we first evaluate the overall performance of three models. Then we give the slot-level performance to look deeply into the relative coverage of the models in each slot and analyze these information. Finally we show the running time of the three models.



a: GeoLife b: Shenzhen Truck

Fig. 4. Relative coverage comparison with various number of selected participants



a: GeoLife $n = 20$ b: GeoLife $n = 40$

c: Shenzhen Truck $n = 100$ d: Shenzhen Truck $n = 200$

Fig. 5. Relative coverage comparison in each slot

Overall Performance Comparison. The overall performance indicates how good the models concerning two datasets. Fig. 4 illustrates the overall relative coverage of three models with the various number of selected participants n. Fig. 4(a) demonstrates model performance in GeoLife, and Fig. 4(b) showcases performance in Shenzhen Truck. We set various n, and the values of n are shown on the x-axis. The height of each bar in the Fig. 4(a) and the Fig. 4(b) represents the average of relative coverage of all slots for the specific model when selecting a fixed number of selected participants. For example, in Fig. 4(a), the purple bar with x-coordinate 30 and y-coordinate 0.64 suggests that when we use DQN-Recruiter to select 30 participants in the FS-MCS tasks based on GeoLife, and the average of relative coverage for all 20 slots is 0.64.

As is shown in Fig. 4(a), the RL-Recruiter outperforms the DQN-Recruiter by achieving 14% higher relative coverage on average, and the CrowdRecruiter by achieving 62% higher relative coverage on average; in Fig. 4(b), it outperforms the DQN-Recruiter by achieving 16% higher relative coverage on average, and the CrowdRecruiter by achieving 21% higher relative coverage on average. These results suggest that RL-Recruiter achieves higher overall performance on two datasets.

Slot-level Performance Comparison. The slot-level performance gives detailed information of the output from the models in each slot. In the Fig. 5(a/b), we display the relative coverage of three models in Geolife with the n set as 20 and 40. In the Fig. 5(c/d), we display the relative coverage of three models in Shenzhen Truck with the n set as 100 and 200. In Fig. 5(a/b/c/d), the x-coordinate is the order of time slots, and the y-coordinate is the relative coverage. Each point in the lines of Fig. 5(a/b/c/d) indicates that under the corresponding slot and participant number n, how good the models are behaving. As there is no data to do the decision model learning on the first day, all models iteratively select participants randomly in the first two slots of GeoLife and the first four slots of Shenzhen Truck. So, the relative coverage of the first two slots in Fig. 5(a/b) and first four slots in Fig. 5(c/d) are similar. We can draw the following two conclusions from Fig. 5. First, RL-Recruiter can achieve higher relative coverage than the baseline methods in most of the slots. Second, RL-Recruiter can achieve a more stable performance than baselines.

We now give intuitive reasons that RL-Recruiter achieves higher performance than two baselines. For CrowdRecuiter, the model consists of two sequential phases. one is mobility prediction, the other is participant selection. Mobility prediction outputs the mobility for each participant respectively, based on the trajectories of the participants in history. Participant selection outputs the optimal participant set according to the mobility of the participants given by the first phase. Under the circumstance of the FS-MCS tasks, the mobile users' mobility profiles are collected from scratch so that it is sparse and irregular in each slot. Therefore the mobility prediction may not be accurate, and this misprediction is further amplified in the second phase when the model tries to select a set of participants based on the mobility prediction, leading to the undesirable result. On the other hand, RL-Recruiter extracts the implicit intelligence by using reinforcement learning, to select participants and avoids explicit prediction on the participants' mobility. Therefore the outcome of the model is improved. For DQN-Recruiter, the neural network maps states and actions to rewards. Our target is to maximize the coverage of the selected participants, so it is reasonable to make the increase of the coverage as the component of the rewards. However, this kind of rewards get a high variance, leading an unstable training process for the DQN which fits the data by tuning the parameters based on the loss. Besides, with the large state space, it is hard to design a neural network to capture the complex relationship between the states, actions, and rewards and a small change in state vectors can lead to huge differences in rewards. And we find that neural network overfits some data during evaluation and impacts its performance. In contrast, RL-Recruiter uses equation 5 to update the value function, and this incremental method achieves a stable performance of the RL-Recruiter. Both of them are approximate solutions, but we find state aggregation more suitable in this problem.

Running Time. We estimate the running time to know the time-efficiency of three models. We use Shenzhen Truck to do the test and calculate the average of time under various settings. The DQN-Recruiter needs about 20 hours to finish 10 slots under the framework of reinforcement learning with the neural network. The CrowdRecruiter spends 4.3 minutes on average. The RL-Recruiter needs 10.8 minutes on average. These models in FS-MCS problem are designed to run offline, thus there is enough time to complete the work, and the running time of three models are all acceptable.

VI. Discussion and Limitation

Various formulations for participant selection problem. In this paper, we first leverage reinforcement learning to address the challenges in the problem of MCS participant selection. With the gradual accumulation of mobility trajectories over time, RL-Recruiter can make a good sequence of participant selection decisions for each sensing slot. In fact, there can be more constraints and goals for the optimization (e.g. energy consumption, participants' bandwidth, various incentive mechanisms, and so on). Moreover, there could be more types of tasks (either homogeneous or heterogeneous) and the selection can be done on either offline or online. In future work, we need to investigate whether the basic idea of RL-Recruiter can be applied to other variants of problem formulations with certain modifications and adjustments.

Joint learning with other factors. As we mentioned in the related work section, there is a new trend to optimize the MCS or general crowdsourcing by exploiting different types of machine learning techniques [23]. In these works, multiple factors of MCS are self-learned from multiple trials over time, such as the participants' expertise [28], required incentives [24], and so forth, which are complementary to RL-Recruiter. In some complex situations, they need to be effectively integrated to jointly optimize the process of MCS. For example, if sensing tasks require a certain expertise to execute and incentives are quoted by participants, we need to establish a joint learning framework by considering these factors with the mobility pattern in RL-Recruiter. Thus, the research on a joint learning framework or a system for MCS participant selection are promising directions in the future. Besides, the proposed lookup table will be large when there is a huge number of participants, which may make it hard to learn good parameters due to the sparsity problem. Thus, we plan to further evaluate and improve our methods with more datasets containing a larger set of participants in future work.

VII. Conclusion

This paper investigated making a good sequence of participant selection decisions in the MCS with the gradual accumulation of mobility trajectories. Specifically, we proposed a novel framework, named RL-Recruiter, to select a suitable set of participants in each sensing slot based on reinforcement learning. The simulations on two real-world mobility datasets demonstrated that RL-Recruiter outperforms the baseline methods in both overall and slot-level coverage. Besides, our approach is not only superior on average performance but also outperforms in a more stable way.

VIII. Acknowledgments

This work was supported by NSFC (National Natural Science Foundation of China) under Grant No. 61872010.

References

[1] R. K. Ganti, F. Ye, and H. Lei. 2011. Mobile crowdsensing: Current state and future challenges. IEEE Communications Magazine, 49: 32-39.

[2] Guo, B., Wang, Z., Yu, Z., et al. (2015). Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. ACM Computing Surveys (CSUR), 48(1), 7.

[3] Wang, J., Wang, L., Wang, Y., Zhang, D., & Kong, L. (2018). Task Allocation in Mobile Crowd Sensing: State of the Art and Future Opportunities. IEEE Internet of Things Journal, 2018, 5 (5), 3747-3757

[4] Sasank Reddy, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using context annotated mobility profiles to recruit data collectors in participatory sensing. In Location and Context Awareness, pages 52–69. Springer, 2009.

[5] Reddy, D. Estrin, and M. Srivastava. Recruitment framework for participatory sensing data collections. In Proceedings of Pervasive, pages 138–155. 2010.

[6] Giuseppe Cardone, Luca Foschini, Paolo Bellavista, Antonio Corradi, Cristian Borcea, Manoop Talasila, and Reza Curtmola. Fostering participation in smart cities: a geo-social crowdsensing platform. Communications Magazine, IEEE, 51(6), 2013.

[7] Zhang, M., Yang, P., Tian, C., & Tang, S. (2015). Quality-aware sensing coverage in budget constrained mobile crowdsensing networks. IEEE Transactions on Vehicular Technology, 1-1.

[8] D Zhang, H Xiong, L Wang, and G Chen. Crowdrecruiter: Selecting participants for piggyback crowdsensing under probabilistic coverage constraint. In Proc. ACM Joint Conference on Pervasive and Ubiquitous Computing (Ubicomp), 2014.

[9] Guo, B., Liu, Y., Wu, W., et al. (2017). Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems. IEEE Transactions on Human-Machine Systems, 47(3), 392-403.

[10] Lane, N. D., Chon, Y., Zhou, L., et al. (2013, November). 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 (p. 7). ACM.

[11] Wang, L., Zhang, D., Wang, Y., et al. (2016). Sparse mobile crowdsensing: challenges and opportunities. IEEE Communications Magazine, 54(7), 161-167.

[12] Z. Song, C. H. Liu, J. Wu, et al. 2014. QoI-Aware Multitask-Oriented Dynamic Participant Selection with Budget Constraints. IEEE Transactions on Vehicular Technology, 63: 4618-4632.

[13] Wang, J., Wang, Y., Zhang, D., et al. (2016). Fine-grained multi-task allocation for participatory sensing with a shared budget. Internet of Things Journal (in press).

[14] Wang, J., Wang, Y., Zhang, D., et al. PSAllocator: Multi-Task Allocation for Participatory Sensing with Sensing Capability Constraints. The ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW 2017).

[15] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In SIGSPATIAL GIS, 2012.

[16] Kazemi, L., Shahabi, C., & Chen, L. (2013, November). Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In Proceedings of the 21st sigspatial international conference on advances in geographic information systems (pp. 314-323). ACM.

[17] Cheng, P., Lian, X., Chen, L., & Shahabi, C. (2017, April). Prediction-Based Task Assignment in Spatial Crowdsourcing. In IEEE 33rd International Conference on Data Engineering, ICDE 2017 (pp. 997-1008). IEEE.

[18] P. Cheng, X. Lian, Z. Chen, and et al. Reliable diversity-based spatial crowdsourcing by moving workers. VLDB.

[19] Q. Liu, T. Abdessalem, H. Wu, Z. Yuan, and S. Bressan. Cost minimization and social fairness for spatial crowdsourcing tasks. In DASFAA, 2016.

[20] Li, Hanshang, Ting Li, and Yu Wang. "Dynamic participant recruitment of mobile crowd sensing for heterogeneous sensing tasks." Mobile Ad Hoc and Sensor Systems (MASS), 2015 IEEE 12th International Conference on. IEEE, 2015.

[21] Li H, Zhu H, Du S, et al. Privacy leakage of location sharing in mobile social networks: Attacks and defense. IEEE Transactions on Dependable and Secure Computing, 2016.

[22] Bilogrevic I, Huguenin K, Agir B, et al. A machine-learning based approach to privacy-aware information-sharing in mobile social networks. Pervasive and Mobile Computing, 2016, 25: 125-142

[23] Wang J, Wang Y, Zhang D, et al. Learning-Assisted Optimization in Mobile Crowd Sensing: A Survey. IEEE Transactions on Industrial Informatics, 2019, 15(1): 15-22.

[24] Karaliopoulos, M., Koutsopoulos, I., & Titsias, M. (2016, July). First learn then earn: Optimizing mobile crowdsensing campaigns through data-driven user profiling. In Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing (pp. 271-280). ACM.

[25] Han, K., Zhang, C., & Luo, J. (2014, June). BLISS: Budget limited robust crowdsensing through online learning. In 2014 Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking (SECON) (pp. 555-563). IEEE.

[26] Li H, Li T, Li F, et al. Cumulative Participant Selection with Switch Costs in Large-Scale Mobile Crowd Sensing. 27th International Conference on Computer Communication and Networks (ICCCN). IEEE, 2018: 1-9.

[27] Liu Y, Liu M. An online learning approach to improving the quality of crowd-sourcing. ACM SIGMETRICS Performance Evaluation Review. ACM, 2015, 43(1): 217-230.

[28] Zhang X, Wu Y, Huang L, et al. Expertise-aware truth analysis and task allocation in mobile crowdsourcing. IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2017: 922-932.

[29] Restuccia F, Ghosh N, Bhattacharjee S, et al. Quality of information in mobile crowdsensing: Survey and research challenges. ACM Transactions on Sensor Networks (TOSN), 2017, 13(4): 34.

[30] Wang J, Wang Y, Zhang D, et al. Multi-task allocation in mobile crowd sensing with individual task quality assurance. IEEE Transactions on Mobile Computing, 2018, 17 (9), 2101-2113.

[31] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529.

[32] Cho, Eunjoon, Seth A. Myers, and Jure Leskovec. "Friendship and mobility: user movement in location-based social networks." Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.

[33] Yu Zheng, Lizhu Zhang, Xing Xie, Wei-Ying Ma. Mining interesting locations and travel sequences from GPS trajectories. In Proceedings of International conference on World Wild Web (WWW 2009), Madrid Spain. ACM Press: 791-800.

[34] Desheng Zhang, Juanjuan Zhao, Fan Zhang, and Tian He. UrbanCPS: a Cyber-Physical System based on Multi-source Big Infrastructure Data for Heterogeneous Model Integration. In the 6th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS'15), 2015.

[35] Lika B, Kolomvatsos K, Hadjiefthymiades S. Facing the cold start problem in recommender systems. Expert Systems with Applications. 2014 Mar 1;41(4):2065-73.

[36] Devarakonda, S., Sevusu, P., Liu, H., Liu, R., Iftode, L., and Nath, B. Real-time air quality monitoring through mobile sensing in metropolitan areas. In UrbComp (2013), 15:1–15:8.

[37] Hasenfratz, D., Saukh, O., Sturzenegger, S., and Thiele, L. Participatory air pollution monitoring using smartphones. In 2nd International Workshop on Mobile Sensing (2012).

[38] https://www.uradmonitor.com/the-worlds-first-mobile-phone-with-air-quality-sensors/