Location-Dependent Task Allocation for Mobile Crowdsensing with Clustering Effect

Xi Tao and Wei Song, Senior Member, IEEE

Abstract—Mobile Crowdsensing (MCS) offers a promising paradigm for large-scale sensing with the rapid growth of mobile smart devices. Compared with traditional sensing methods, MCS is more effective and efficient in energy and cost. Task allocation is a key problem in MCS, which has a significant impact on the performance. It is challenging to design a generic solution to the task allocation problem because MCS applications typically consider distinct targets under specific constraints. However, there are many common interests such as data quality, budget, and energy consumption. In this paper, we analyze and formulate the task allocation problem from two perspectives, respectively. First, we focus on data quality and propose a genetic algorithm (GA) to maximize data quality. Then, we take the profit of workers into account and propose a detective algorithm (DA) to improve the profit. In the GA-based solution, only the platform is able to decide the task assignment. However, in the DA-based solution, the workers are allowed to determine and submit their task sets to the platform, which just needs to make a selection from these task sets. In addition, we consider the clustering effect of tasks and the influence caused by different geographic distributions of tasks. To evaluate the performance of the proposed solutions, extensive simulations are conducted. The results demonstrate that our proposed solutions outperform the baseline algorithm and there is a trade-off between the data quality and the profit of workers.

Index Terms—Mobile crowdsensing, task allocation, genetic algorithm, clustering effect.

I. INTRODUCTION

The evolving Internet of Things (IoT) allows objects to be controlled remotely across the existing networks with the growth of embedded devices [1]. These sensor-rich devices enable plenty of IoT applications in real life (e.g., environment, transportation, and healthcare). In general, data of IoT applications flow in two opposite directions from servers to users (i.e., data dissemination) or users to servers (i.e., data sensing). Data dissemination provides methods of distributing data to end users [2,3]. Conversely, data sensing acquires information from users or sensors. With the explosive growth of mobile smart devices, researchers start to explore the possibility of leveraging crowd intelligence in IoT applications [4]. For example, crowdsourcing is a paradigm that allows individuals to receive data or services from a large group of users [5]. In recent years, sensor-equipped mobile smartphones are becoming indispensable in people’s daily life, which provides ubiquitous sensors (e.g., GPS, microphone, and camera) as available resources for IoT applications [6].

Mobile crowdsensing (MCS), a term coined by Ganti et al. [7], offers a new method to sense and share data among users or communities. There are four main components in MCS, i.e., the platform, requesters, tasks, and workers. Fig. 1 shows that tasks are posted to the platform by requesters and further assigned to workers, while sensed data is collected from workers to the platform and requesters. Due to the involvement of humans, MCS is also termed people-centric sensing, which is further classified into personal sensing, public sensing, or social sensing based on the sensing scale [8]. On the other hand, it is more common to classify MCS into participatory sensing and opportunistic sensing according to the degree of people’s involvement [9,10]. Participatory sensing encourages users with mobile smart devices to gather and share local knowledge. In participatory sensing, workers actively search for and carry out their qualified sensing tasks to acquire the incentive rewards. In contrast, workers in opportunistic sensing are not aware of the participation. The activities of sensing and uploading are autonomous if workers pass by the task locations and the requirements for these activities are met. The distinct characteristics of participatory sensing and opportunistic sensing result in significant differences with respect to the behaviour of workers and completion of tasks.

In addition, these different sensing modes of MCS can serve appropriate types of tasks. Tasks in MCS can be deadline-sensitive or delay-tolerant according to the urgency of uploading. For deadline-sensitive tasks, the sensed data samples need to be uploaded immediately after sensing due to a explicit deadline of uploading, which makes it strict to recruit a sufficient number of workers. For instance, drivers need the up-to-date traffic condition at an intersection. As a consequence, participatory sensing is more suitable for deadline-sensitive tasks than opportunistic sensing. This is because workers in

Fig. 1. Structure of MCS.
participatory sensing are able to intentionally move to the task locations and finish the sensing and uploading activities on time, which leads to a larger coverage and complete ratio of tasks. However, participatory sensing is costly in budget since workers are more willing to travel around and upload data samples if a sufficient incentive is provided [11]. On the other hand, delay-tolerant tasks allow a certain delay for the uploading of data samples after sensing, which enables more cost-efficient methods. Workers in opportunistic sensing can finish the sensing and uploading without intentional movement and these tasks are assigned based on workers’ historical trajectories. Although the collection of historical records entails certain preparatory work, opportunistic sensing saves energy for workers and budget for requesters. Nevertheless, it is hard to guarantee the performance of opportunistic sensing as it can be difficult to predict people’s mobility patterns only based on limited historical data.

In order to appropriately assign different types of tasks in MCS, a key challenge is to solve the task allocation problem. On one hand, it is not possible to find a generic solution that exhausts and satisfies all requirements and constraints of tasks. Thus, it is necessary to use analytical methods (e.g., 4W1H [12]) to identify the specific requirements and constraints of the task allocation problem in a certain scenario. On the other hand, requesters and workers are not always consistent in interests. For example, requesters are concerned with data quality, while workers expect a satisfactory profit. Obviously, data quality can be improved by increasing the budget and reward of tasks to attract and recruit workers who are able to provide high-quality data samples. Meanwhile, workers can receive a competitive profit. However, in budget-constrained situations, the targets of requesters and workers are more conflicted. Unfortunately, budget is a precious and limited resource in most MCS applications.

The studies on MCS have resulted in a variety of applications such as environment monitoring, traffic planning, and healthcare [13]. For instance, the Ear-Phone system offers a novel MCS application for urban noise pollution monitoring based on participatory sensing [14]. Ear-Phone is implemented on smartphones to collect noise readings and generate a noise map to identify noise pollution levels in a certain area. NoiseTube is another smartphone-based MCS system that maps and monitors noise pollution [15]. Besides a mobile application, NoiseTube has a Web-based community memory system on the server to explore, visualize, and analyze the noise pollution data. MCS is also used for traffic planning such as SignalGuru for traffic signal schedule [16]. SignalGuru uses opportunistic sensing to receive traffic signals automatically captured by mobile phones’ cameras and predicts future traffic signal schedule via collaborative learning of the patterns from the collected data.

In this paper, we take an attempt to address a location-dependent task allocation problem in MCS. First, the geographic distribution of tasks is an important issue, which should be considered in the task allocation problem. Normally, the sensing tasks can be located as a uniform distribution or a clustering distribution. For example, if the requesters need a map of air pollution level in a specific area, it is better to place tasks uniformly in this area to monitor the pollution. On the contrary, it is common to find a clustering distribution of sensing tasks when the requesters aim to know the parking space near a shopping mall during the busy hours.

We focus on solving the allocation problem with the clustering tasks, but we also investigate the effect of different geographic distributions. Second, when it comes to the data quality of returned data samples, we distinguish multiple data samples for the same task with respect to their values considering the data redundancy effect. For a specific task, the first data sample provides the largest value, while the values of the subsequent samples decrease continuously. Therefore, the total value of data samples collected for the task, i.e., the overall data quality of the task, presents a diminishing marginal increase. Meanwhile, the task reward is reducing corresponding to the decreasing value of data samples, which causes competition and conflicts among workers.

Specifically, our main contributions are three folds.

- We study the task allocation problem taking geographic distributions of tasks into account, especially the clustering tasks. In addition, the effect of geographic distributions on performance is analyzed with specific examples.
- We formulate the task allocation problem from two perspectives. The diminishing marginal increase of data quality and the competition of workers are considered in our formulation.
- We propose efficient solutions to the formulated task allocation problem. A GA-based solution is proposed to maximize the data quality and a fast DA-based solution is proposed to improve the profit of workers.

The remainder of this paper is organized as follows. The related work is presented in Section II. Section III gives the system model and problem formulation. We present the details of the proposed solutions in Section IV. The simulation results are shown in Section V. Section VI concludes this paper.

II. RELATED WORK

In this section, we review the related work on task allocation for MCS. For ease of comparison, we classify the existing solutions into three groups, i.e., participatory solutions, opportunistic solutions, and hybrid solutions.

Participatory Solutions - Participatory solutions are based on participatory sensing that is efficient to plan valid paths for workers under various geographic requirements (e.g., coverage, and travelling distance). He et al. [17,18] proposed a solution for task allocation to maximize the reward of the platform under the travelling distance limits for workers. In this solution, each task needs to receive multiple data samples to guarantee its data quality, which certainly causes the data redundancy problem. In order to suppress data redundancy, a maximum number of data samples are allowed for each task. An efficient local-ratio-based algorithm is used in their solution. The simulation results are based on a random distribution of locations.

As known, the ability of participatory sensing to achieve a high task complete ratio or coverage is at the cost of large rewards that are provided to workers for sensing and uploading.
activities. Therefore, the budget is a key issue in participatory solutions. Wang et al. [19] considered the multitask allocation problem under a shared budget. In each sensing cycle, a subset of tasks is assigned to each worker to maximize the overall data quality, which is defined as a weighted sum of the data quality indices. The total cost of all workers cannot exceed a shared budget. Similarly, Zhou et al. [20] and Zhang et al. [21] also considered the budget constraint, although different targets were proposed. In [20], the authors aimed at maximizing the data quality with a budget of total travelling distance. An online learning approach was proposed to achieve the goal. In [21], the quality is defined as the coverage, while the target is to maximize the coverage quality under the budget. An approximation algorithm was utilized and evaluated in random-walk-based simulation.

Social surplus is another important performance benchmark for task allocation in MCS. In [22], Cheung et al. studied task allocation for time-sensitive and location-dependent tasks, aiming at maximizing the social surplus, which is defined as users’ total rewards minus total movement costs. A centralized task allocation problem is formulated and proved to be NP-hard. A low-complexity greedy algorithm is proposed to find an approximate solution by sorting tasks in execution time and allocating one user to each task in a greedy manner. In addition, this work formulates a distributed task selection game, which allows every user to choose a task-time route that maximizes its payoff. Both the distributed and centralized solutions are shown to perform efficiently in terms of user payoff and fairness, and task coverage.

Opportunistic Solutions - Opportunistic solutions take advantage of opportunistic sensing to find energy and cost efficient solutions by reusing available resources for sensing and uploading activities. For example, piggyback crowdsensing is an approach that collects data by exploiting application opportunities of smartphones to reduce the energy cost [23]. Based on piggyback crowdsensing, a task allocation framework called CrowdTasker was designed to maximize the coverage quality under a budget constraint [24]. In CrowdTasker, a number of workers are selected from the worker pool and then assigned certain tasks. The authors proposed a two-phase solution that considers the mobility and historical records of workers. By analyzing the pattern of workers’ mobility, CrowdTasker predicts the opportunities of encounters between workers and tasks, which enables workers to conduct sensing and upload the data samples opportunistically in the near future. In the extended framework named iCrowd [25], a novel spatial-temporal coverage metric is further defined in terms of $k$-depth coverage. The workers are selected in iCrowd based on the mobility prediction to achieve two goals, i.e., maximizing $k$-depth coverage under budget constraint and minimizing overall payment under $k$-depth coverage constraint. A greedy search method was proposed for these two dual problems.

Similarly, EEMC [26] and EMC$^3$ [27] were also designed based on the prediction of workers’ mobility. However, the target is to minimize the number of recruited workers instead of maximizing the coverage quality. Based on the mobility prediction, the probability that potential data collectors appear in a specific cycle or area is evaluated. Then, tasks are assigned to appropriate workers so that the number of recruited workers is as small as possible.

There are also some works focusing on data cost such as ecoSense [28] and effSense [29]. In these solutions, the workers are divided into two groups according to their different data plans - unlimited data plan and pay-as-you-go plan. The target is to minimize the uploading cost of delay-tolerant tasks by relaying data samples among workers. Workers in different groups choose their own strategies and they also collaborate to achieve a better overall system performance. For example, workers with unlimited data plan can act as relays for those who are lack of data uploading capacity in some situations.

To address high dynamics of tasks and workers, some online solutions have also been studied. For example, a quality-aware self-organized framework, Crowd Forging, is proposed in [30], in which a mobile task requester can recruit in real time a crowd of opportunistic encountered mobile workers. Taking into account various factors for tasks and workers (e.g., workers’ arrivals and abilities), the authors formulated an online multiple stopping problem for worker recruitment to maximize the expected sum of service quality. An optimal policy is derived through dynamic programming.

Hybrid Solutions - Both participatory sensing and opportunistic sensing have their respective strengths. For example, participatory sensing can achieve a large coverage, while opportunistic sensing saves energy and budget. There is no doubt that a generic framework of task allocation should be able to deal with different types of tasks, varied constraints, and specific targets. Therefore, it is promising to combine participatory sensing and opportunistic sensing together. There have been some attempts for such hybrid solutions. ActiveCrowd is a worker selection framework for both time-sensitive tasks and delay-tolerant tasks [31]. The targets of task allocation vary with different types of tasks. For allocation of time-sensitive tasks, the target is to minimize the travelling distance. For allocation of delay-tolerant tasks, the target is to minimize the number of recruited workers. In order to solve the two different task allocation problems, two greedy-enhanced genetic algorithms were proposed with participatory sensing for time-sensitive tasks and opportunistic sensing for delay-tolerant tasks.

There are two modes of task allocation when tasks are location-dependent. One mode is worker-centric, in which the platform publishes the information of tasks. Then workers make their own decisions without coordination with other workers and the platform. The worker-centric mode can protect the privacy of workers, since workers do not need to reveal their locations to the platform. The other mode is platform-centric. Workers who are willing to take part in the sensing activities send their locations to the platform and the platform assigns tasks to every worker. The platform-centric mode has the information of all workers so that it is possible for the platform to apply a globally optimal allocation. Kazemi and Shahabi compared these two different modes and analyzed the advantages and drawbacks of both modes [32]. A maximum task assignment problem is defined and formulated as a network flow problem. Three different strategies are proposed to solve the problem, i.e., the greedy strategy, the least location
TABLE I
NOTATION DEFINITIONS.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Set of tasks</td>
</tr>
<tr>
<td>$t_j$</td>
<td>Task $j$</td>
</tr>
<tr>
<td>$W$</td>
<td>Set of workers</td>
</tr>
<tr>
<td>$w_i$</td>
<td>Worker $i$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>Path of worker $i$</td>
</tr>
<tr>
<td>$l_i$</td>
<td>Maximum travelling distance of worker $i$</td>
</tr>
<tr>
<td>$n_j$</td>
<td>Number of data samples task $j$ receives</td>
</tr>
<tr>
<td>$d(P_i)$</td>
<td>Distance function for the length of $P_i$</td>
</tr>
<tr>
<td>$c(P_i)$</td>
<td>Cost function for path $P_i$</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>Weight of sensing cost for worker $i$</td>
</tr>
<tr>
<td>$\beta_i$</td>
<td>Weight of travelling cost for worker $i$</td>
</tr>
<tr>
<td>$r_j(n_j)$</td>
<td>Reward function for task $j$</td>
</tr>
<tr>
<td>$\gamma_j$</td>
<td>Initial reward for task $j$</td>
</tr>
<tr>
<td>$\lambda_j$</td>
<td>Reward decreasing rate for task $j$</td>
</tr>
<tr>
<td>$v_j(n_j)$</td>
<td>Value function for task $j$</td>
</tr>
<tr>
<td>$\delta_j$</td>
<td>Factor of importance for value of task $j$</td>
</tr>
<tr>
<td>$\theta_j$</td>
<td>Value increasing rate for task $j$</td>
</tr>
</tbody>
</table>

entropy priority strategy, and the nearest neighbour priority strategy.

III. SYSTEM MODEL AND PROBLEM FORMULATIONS
In this section, we first give the system model for task allocation in MCS. Then, we formulate the task allocation problem from two perspectives of requesters and workers. At last, we analyze the computational hardness of our formulated problems. For easy reference, we list the important notations used in this paper in Table I.

A. System Model

1) Locations of Tasks and Workers: In a MCS framework or application, there are a set of sensing tasks that are posted by requesters and a set of workers, denoted by $T = \{t_1, t_2, ..., t_m\}$ and $W = \{w_1, w_2, ..., w_n\}$, respectively. The geographic locations of sensing tasks are provided by requesters and the locations of workers can be obtained via positioning measurement (e.g., GPS). Here, we assume that the initial locations of workers are uniformly distributed in a specific area. In contrast, the location distribution of tasks presents a clustering effect. These task clusters can be simulated by applying a cluster method (e.g., the $k$-means method) to the uniform distribution. To illustrate the MCS scenario under study, Fig. 2 shows an example with the spatial distributions of tasks and workers. The task allocation problem is to determine the sensing tasks assigned to each worker and the worker’s travelling path to carry out the assigned tasks. For each worker $w_i \in W$, let $P_i$ denote its travelling path that includes a sequence of assigned tasks. Therefore, the solution to the task allocation problem can be represented by $\{P_i | \forall w_i \in W\}$.

2) Maximum Travelling Distance: Generally speaking, the resources (e.g., energy) of workers are not infinite. The most important resource of workers in our MCS scenario is the ability to travel around for sensing and uploading activities, i.e., the maximum travelling distance. For each worker $w_i \in W$, let $l_i$ denote its maximum travelling distance. Hence, a feasible task assignment to worker $w_i$ should be a travelling path $P_i$ with a distance less than the maximum travelling distance $l_i$. The travelling distance over path $P_i$ is denoted by $d(P_i)$, so a feasible path $P_i$ must meet the condition $d(P_i) \leq l_i$.

3) Cost and Reward: Obviously, data sensing and uploading involve a certain cost to workers. We assume that the total cost of workers is divided into two components, i.e., sensing cost and travelling cost. For each worker $w_i \in W$ and its corresponding travelling path $P_i$, the cost is evaluated by:

$$c_i(P_i) = \alpha_i \times |P_{-i}| + \beta_i \times d(P_i).$$ (1)

In (1), the first term in the right-hand side is the sensing cost, which is proportional to the number of tasks in path $P_i$. Here, $|P_{-i}|$ indicates the number of vertices in path $P_i$ excluding the first node, which is the initial location of worker $w_i$. This characterization of the sensing cost implies that every task along path $P_i$ involves the same sensing cost to the worker. Nonetheless, it would be easy to adapt this term to consider heterogeneous sensing costs of tasks. The second term in the right-hand side refers to the travelling cost that depends on the travelling distance $d(P_i)$ of worker $w_i$. Last, $\alpha_i$ and $\beta_i$ define the weights for sensing cost and travelling cost, respectively.

Due to the cost of sensing and uploading activities, it is indispensable to provide a reward for workers to compensate for their cost and stimulate their participation. As mentioned in Section I, the reward for a data sample decreases when more data samples have been collected because of the diminishing marginal increase of data quality. From the perspective of data quality, data redundancy can enhance data accuracy to a certain degree. However, too many data samples for the same task inevitably cause a waste of budget. For each task $t_j \in T$, let $n_j$ denote its current number of returned data samples. The reward of one data sample associated with task $t_j$ is characterized by:

$$r_j(n_j) = \gamma_j \times e^{-\lambda_j n_j}$$ (2)

where $\gamma_j$ is the initial reward for task $t_j$ and $\lambda_j$ is a factor to control the effect of diminishing marginal increase. A larger $\lambda_j$ means that the reward of data samples reduces more quickly.
Before computing the new reward \( r_j(n_j) \), the current number of data samples \( n_j \) needs to be updated by:

\[
n_j(\bigcup_{i=1}^{m} P_i) = \sum_{i=1}^{m} 1(t_j \in P_i). \tag{3}
\]

In (3), \( 1(\cdot) \) is the indicator function, which equals 1 when the condition in the function argument is satisfied. Hence, (3) gives the current number of data samples received for task \( t_j \) based on the current travelling paths of workers.

### B. Problem Formulation

Based on the system model defined in Section III-A, we formulate the task allocation problem from two perspectives according to the role of workers including a requester-centric problem and a worker-centric problem. In the requester-centric problem, the task assignment is decided by requesters or the platform. In the following, we assume that the platform makes the decision on behalf of requesters. Workers are just followers to accept the assignment and they are not allowed to plan their own paths. In contrast, the worker-centric problem considers workers as decision makers. First, workers plan their own paths to improve their profit and submit their paths to the platform. Then, the platform makes a selection of these submitted plans.

1) Requester-centric Problem: As mentioned in Section I, each data sample has a certain value, which contributes to the overall data quality of a task, i.e., the total summation of the values of all data samples collected for this task. Although data redundancy requires a large budget, the data quality is improved by multiple data samples from distinct workers, which ensures independence of these data samples and prevents the data quality from possible corruptions or errors. Thus, the data quality for task \( t_j \) is increased with the growing of returned data samples. However, data values present a diminishing marginal increase because of the unequal values of data samples. Based on the above considerations, we model the value function for task \( t_j \) as follows:

\[
v_j(n_j) = \delta_j \times n_j^{\theta_j}. \tag{4}
\]

In (4), \( \delta_j \) is a measure for the importance of task \( t_j \) and a larger \( \delta_j \) means that task \( t_j \) is more valuable than others. In addition, \( 0 \leq \theta_j < 1 \), is an indicator of the increasing rate of value. A small \( \theta_j \) means that data redundancy helps little in improving the data quality.

In order to incentivize participation, each worker \( w_i \in W \) is willing to accept a task assignment only if its cost is not higher than the reward, i.e., \( c_i(P_i) \leq \sum_{t_j \in P_i} r_j(n_j) \). In addition, the total reward of all data samples cannot exceed a predefined budget, denoted by \( b \). Further taking into account the maximum travelling distance of workers, we can formulate the requester-centric problem in (5). Here, the objective function in (5a) is to maximize the total value of all tasks in data quality. Constraint (5b) defines the travelling distance limit for each worker; (5c) gives the incentive constraint that the cost of each worker cannot exceed its reward; and (5d) indicates that

\[
\begin{align*}
\max. & \sum_{j=1}^{m} v_j(n_j) \tag{5a} \\
\text{s.t.} & d(P_i) \leq b, \forall w_i \in W \tag{5b} \\
& c_i(P_i) \leq \sum_{t_j \in P_i} r_j(n_j), \forall w_i \in W \tag{5c} \\
& \sum_{i=1}^{n} \sum_{t_j \in P_i} r_j(n_j) \leq b. \tag{5d}
\end{align*}
\]

2) Worker-centric Problem: Next, we consider the task allocation problem from another perspective. Workers pay attention to their profit rather than the value of data quality. The condition that the reward is not less than the cost is not strong enough to attract participation of workers. Therefore, we assume that, in some situations, workers are permitted to plan the travelling paths to improve their profit. Then the platform only makes a selection of these paths to maximize the value of data quality.

To solve the competition problem of workers, we introduce the concept of cycles. Fig. 3 shows an example to illustrate the function of cycles. An individual worker may receive different rewards even for taking the same tasks because of the decreasing reward in (2). Thus, the order of taking the same tasks by different workers matters. In the example in Fig. 3, worker #1 is first assigned to collect the first data samples for task #1 and task #2, and it receives a total reward of 6. In contrast, worker #2 who is assigned later only receives a total reward of 4 as it collects the second data samples for tasks #1 and #2. This discrepancy issue can be solved if we cut the whole sensing period into small cycles and declare the reward of every task at the beginning of each cycle. Each task’s reward keeps the same in an entire cycle. The reward for task #1 is declared to be 4 when cycle 1 begins, while the reward for task #2 is given as 2 when cycle 2 begins. Thus both workers #1 and #2 who carry out the same tasks receive an equal reward of 6 regardless of the assignment order. As shown in this example, the task rewards in the worker-centric problem are also determined according to (2) as in the requester-centric problem. Nonetheless, In the requester-centric problem, the number of data samples of each task is renewed immediately after a new data sample is received and thus the task rewards are also updated immediately. In contrast, the task rewards in the worker-centric problem are updated at the end of each cycle. This can improve fairness among workers and solve the competition problem illustrated in Fig. 3.
Generally, at the beginning of cycle $k$, for each task $t_j \in T$, the reward for each data sample $r_j(n^k_j)$ is determined according to the current number of data samples $n^k_j$. For each worker $w_i \in W$, the travelling path $P^k_i$ in this cycle is planned by worker $w_i$ to maximize its profit. The travelling distance of path $P^k_i$ is limited in two aspects. One limit is the remaining travelling distance, which equals the maximum travelling distance minus the distance travelled so far, denoted by $\hat{l}_i$. The other limit is the length of cycles, which is denoted by $h$. For each worker $w_i \in W$, its moving speed is denoted by $f_i$ and the maximum travelling distance in this cycle cannot be larger than $h \cdot f_i$. Since each worker plans its own path, there are $n$ path planning problems and the problem for worker $w_i$ is formulated as follows:

\[
\text{max. } \sum_{t_j \in P^k_i} \left[ r_j(n^k_j) - c_i(P^k_i) \right] \quad (6a)
\]

\[
\text{s.t. } d(P^k_i) \leq \min\{\hat{l}_i, h \cdot f_i\}, \forall w_i \in W. \quad (6b)
\]

Here, objective function in (6a) is to maximize the profit for worker $w_i$ in cycle $k$. Constraint (6b) defines the travelling distance limit for worker $w_i$ in cycle $k$.

Workers submit their paths to the platform at the end of each cycle. Then the platform selects a subset of submitted paths to maximize the value of data samples for the best data quality. The subset of selected paths is denoted by $S^k$ and the selection needs to satisfy the budget constraint, i.e., the total reward for all data samples in cycle $k$ cannot exceed the remaining budget denoted by $\hat{b}$. The selection problem is formulated as follows:

\[
\text{max. } \sum_{P^k_i \in S^k} \sum_{t_j \in P^k_i} v_j(n^k_{j}^+) \quad (7a)
\]

\[
\text{s.t. } \sum_{P^k_i \in S^k} \sum_{t_j \in P^k_i} r_j(n^k_{j}) \leq \hat{b}. \quad (7b)
\]

Here, $v_j(n^k_{j}^+)$ is set to $n^k_{j}$ at the beginning, but it is increased by one whenever one more sample is added by selecting a new set that contains task $t_j$. The objective function in (7a) is to maximize the value of data quality in cycle $k$. Constraint (7b) defines the budget limit for task assignment in cycle $k$.

The cycle $(k+1)$ comes after cycle $k$. For each task $t_j \in T$, the number of data samples $n^k_j$ is updated to $n^{k+1}_j$ and the reward $r_j(n^{k+1}_j)$ is renewed. The remaining budget $\hat{b}$ and the remaining travelling distance $\hat{l}_i$ of each worker $w_i \in W$ are updated for cycle $(k+1)$. The path plan and selection are repeated iteratively cycle by cycle until no worker can carry out sensing and uploading without violating the distance or budget constraints.

C. Computational Hardness

In the following, we analyze the computational hardness of our formulated problems and provide the proofs of the computational hardness.

**Theorem 1.** The requester-centric problem in (5) is NP-hard.

**Proof.** First, we consider the decision form of problem (5), given by

\[
\text{find } P_i, \forall w_i \in W, \text{ with } \sum_{j=1}^{m} v_j(n_j) \geq \rho \quad (8a)
\]

\[
\text{s.t. } d(P_i) \leq \hat{l}_i, \forall w_i \in W \quad (8b)
\]

\[
c_i(P_i) \leq \sum_{j \in P_i} r_j(n_j), \forall w_i \in W \quad (8c)
\]

\[
\sum_{i=1}^{n} \sum_{j \in P_i} r_j(n_j) \leq b. \quad (8d)
\]

As seen, the decision form is to decide whether there exist paths $\{P_i, \forall w_i \in W\}$ that achieve an objective value not less than $\rho$, where $\rho$ is a given threshold. It takes polynomial time at most $O(nm)$ to verify constraints (8b), (8c), and (8d), or to compare the objective value with the threshold $\rho$. Therefore, problem (8) is NP, since it needs polynomial time to obtain a yes-or-no answer to the decision problem.

Next, we prove that problem (8) is NP-complete, by reducing a known NP-complete problem, i.e., the multi-constrained path problem (MCP), to an instance of problem (8). The decision form of the MCP is defined as follows. Let $G = (V, E)$ be a graph with weight function $\omega : E \mapsto R$ and length function $\ell : E \mapsto R$. The MCP determines whether there is a path from vertex $v_1$ to $v_2$, where $v_1, v_2 \in V$, with weight at most $\Omega$ and length at most $L$. It is shown in [33] that the MCP is NP-complete.

In the following, we construct an instance of problem (8) to solve the MCP. Fig. 4 shows an example illustrating the construction. Here, graph $G$ is an almost complete graph that connects a single worker $w_i \in W$, all tasks $\forall t_j \in T$, and a virtual vertex $\mu$. There is a bidirectional edge between any two vertices, except that there are only unidirectional outgoing edges with the vertex corresponding to worker $w_i$, and unidirectional incoming edges with the virtual vertex $\mu$. Vertex $v_1$ is mapped to the initial location of worker $w_i$, while vertex $v_2$ is mapped to vertex $\mu$. Then, for any edge $e \in E$, length $\ell_e$ is the physical distance between the two ending vertices of edge $e$, except that length $\ell_e$ for incoming edges with vertex $\mu$ is set to zero. The weight of edge $e$ is defined as

\[
\omega_e = \begin{cases} 
(\alpha_i + \beta_i \ell_e) - r_j, & \text{if } t_j \neq \mu \\
\Omega, & \text{if } t_j = \mu
\end{cases} \quad (9)
\]

where task $t_j$ is the tail vertex of edge $e$. As seen, the weight in (9) basically defines the difference of the cost and the reward.
for worker $w_i$ to perform task $t_j$ over edge $e$. Thus, a path that satisfies constraint (8c) is a path with a total weight not more than $\Omega$. Similarly, constraint (8b) is translated to a path with a total length not more than $L = \ell_v$, where $\ell_v$ is the travelling limit of user $w_i$. Last, the instance of problem (8) has a positive infinite budget in (8d) and an objective function with threshold $\rho = 0$ for (8a).

Therefore, if there exists a feasible solution to this instance of problem (8), the answer is yes. Conversely, the answer is no. Thus, the MCP is known to be NP-complete, the decision form of the requester-centric problem in (8) is also NP-complete. Hence, the corresponding optimization form defined in (5) is NP-hard.

**Theorem 2.** The worker-centric problem including (6) and (7) is NP-hard.

*Proof.* First, we consider the decision form of problem (6), given by

$$\begin{align*}
\text{find} & \quad P_i^k, \forall w_i \in W, \text{ with } \sum_{t_j \in P_i^k} [p_j(n_j^k) - c_t(P_i^k)] \geq \xi \\
\text{s.t.} & \quad d(P_i^k) \leq \min\{\hat{\ell}_v, h \cdot f_i\}, \forall w_i \in W.
\end{align*}$$

(10a)

(10b)

Obviously, problem (10) is NP, because we can obtain a yes-or-no answer within polynomial time $O(n^m)$.

In addition, we also prove problem (10) to be NP-complete, by reducing the MCP to an instance of problem (10). Consider an instance of problem (10). Then, we map graph $G$ in the MCP to an almost complete graph as illustrated in Fig. 4 that connects a single worker $w_i \in W$, all tasks $\forall t_j \in T$, and a virtual vertex $\mu$. The MCP determines whether there is a path from vertex $v_1$ to $v_2$, where $v_1, v_2 \in V$, with weight at most $\Omega$ and length at most $L$.

Similar to the proof for Theorem 1, we map vertex $v_1$ to the initial location of worker $w_i$, while vertex $v_2$ is mapped to virtual vertex $\mu$. The length of any edge $e \in E$, $\ell_e$, is the physical distance between the two ending vertices of edge $e$, except that length $\ell_e$ for incoming edges with vertex $\mu$ is set to zero. Also, the weight of any edge $e$ is defined as in (9).

Then, constraint (10b) is equivalent to require that a path’s total length be not more than $L = \min\{\hat{\ell}_v, h \cdot f_i\}$. The condition in the objective function (10a) corresponds to the constraint that a path’s total weight is not more than $\Omega = -\xi$. Note that the summation term in (10a) is the negation of the total weight as defined in (9). As seen, with this constructed instance of problem (10), a yes answer implies that there exists a feasible solution to the MCP. Conversely, the answer is no. Therefore, problem (10) is NP-complete and its corresponding optimization form in (6) is NP-hard.

For the worker-centric problem, there is another subproblem in (7) for each cycle. Next, we show that a special case of the selection problem in (7) is NP-hard. Therefore, the generalized problem (7) is also NP-hard.

Consider an NP-hard budgeted maximum coverage problem (BMCP). In the BMCP, the input is a number of sets of elements, where each element has a weight and each set has a cost. The output is to select certain sets from the input so that the total weight of the covered elements in the selected sets is maximized, while the total cost of selected sets is not more than a given budget.

Consider a special instance of the selection problem in (7), where $\theta_j = 0$, i.e., $v_j = \delta_j, \forall t_j \in T$. This special instance of problem (7) is in fact an NP-hard BMCP. In this instance, we can view each individual data sample as an element. Thus, each path $P_i^k$ submitted by worker $\forall w_i \in W$ corresponds to an input set. The reward to this worker is considered as the cost of the set, while the value of each data sample contained in the set is viewed as the weight of the element. Therefore, problem (7) is equivalent to selecting the sets that give the highest total weight subject to the budget constraint. Hence, this instance of problem (7) is a BMCP. The generalized problem in (7) is thus NP-hard.

In summary, the worker-centric problem including (6) and (7) is an NP-hard problem.

**IV. PROPOSED TASK ALLOCATION SOLUTIONS**

In this section, we propose two solutions to the task allocation problems formulated in Section III-A, i.e., the requester-centric problem and worker-centric problem. For the requester-centric problem, a variant genetic algorithm (GA) is proposed to maximize the value of data samples to improve data quality. For the worker-centric problem, we design a detective algorithm (DA) to maximize the profit of workers.

**A. GA-based Solution**

As analyzed in Section III-B, the search space for the solutions to problem (5) is huge. Hence, traditional combinatorial algorithms are inefficient to search for the best solution. To achieve a satisfactory performance, it is essential to design an algorithm with “global vision”. Fortunately, evolutionary algorithms provide efficient solutions for combinatorial problems like problem (5). Therefore, we propose a solution based on a variant genetic algorithm (GA), which can take all the information into account when planning the travelling paths for workers.

The GA is an evolutionary computing technique that is inspired by natural selection in biological evolution. The GA starts from an initial generation with a certain population of individual candidates and evolves iteratively toward an optimal solution. Each candidate has a set of properties (chromosomes or genotype), which can be altered and mutated. The initial generation is randomly generated in most instances. Then, the fitness (usually defined as the value of objective function) of every individual candidate is evaluated in current generation. Afterwards, the GA creates a new generation by three main types of genetic operators, i.e., selection, crossover, and mutation, based on the fitnesses of candidates. According to the selection rule, the candidates with higher fitness are more likely to be chosen as “parents” to breed “children” for the next generation. Crossover and mutation are two commonly used genetic operators to produce a child from a pair of parents selected above. Crossover combines the characteristics of the parent candidates to form a child candidate, while mutation
Algorithm 1: A GA-based solution for task allocation.

**Input:** \( T \) (set of tasks), \( W \) (set of workers), \( loc_t \) (locations of tasks), \( loc_w \) (initial locations of workers), \( b \) (budget), \( \{l_i|v_{wi} \in W\} \) (maximum travelling distances of workers), \( pop \) (size of population), \( g_{\text{max}} \) (maximum number of generations)

**Output:** \( \{P_i|v_{wi} \in W\} \) (travelling paths of workers)

1. \( g \leftarrow 1 \) // Number of generations
2. Initialize task subsets of \( pop \) workers in first generation
3. Plan travelling paths for workers in first generation
4. Check feasibility and modify all infeasible paths to be feasible
5. Calculate fitness of each candidate in first generation
6. Record solution with highest fitness as \( \{P_i|v_{wi} \in W\} \)
7. **while** \( g < g_{\text{max}} \) **do** // Maximum number of generations is not reached
   8. \( g \leftarrow g + 1 \)
   9. Apply roulette wheel selection to select parents for next generation
   10. Apply crossover to selected parents to generate children
   11. Apply mutation to newly generated children
   12. Design travelling paths for new generation
   13. Check feasibility and modify all infeasible paths to be feasible
   14. Calculate fitnesses and update the best solution
8. **return** \( \{P_i|v_{wi} \in W\} \)

introduces random changes to the genomes that a child inherits from the parents. The evolution terminates when a maximum number of generations have been produced, or the best fitness has reached a satisfactory level.

An individual candidate solution for problem (5) is a set of feasible travelling paths. As it is hard to perform the operations of crossover and mutation directly on the travelling paths of workers, we employ a variant GA. An individual candidate solution is divided into two steps. The first step selects a subset of tasks for each worker, while the second step determines a feasible travelling path based on this subset of tasks. Alg. 1 shows the details of the proposed GA-based solution. Here, the fitness is defined as the value of objective function in problem (5). The GA-based solution ends when the number of generations reaches a given threshold.

In the initial generation, a population of individual candidate solutions are generated as follows. First, for each worker, a subset of tasks are randomly selected. Then, a travelling path is planned by taking the nearest task in the subset until all the tasks in the subset are covered. It is possible that the length of planned path exceeds the maximum travelling distance, which makes the path infeasible. To transform an infeasible path to be feasible, we keep removing the furthest task in the task subset until the travelling distance satisfies the requirement. If all paths of workers are feasible, an individual candidate solution is formed and its fitness can be evaluated. The first generation contains a population of such individual candidate solutions that are constructed as above.

In the subsequent generations, roulette wheel selection (also known as fitness proportionate selection) is applied as the selection rule. Each individual candidate solution is associated with a probability to be selected, which is a proportion of its own fitness over the summation of all fitnesses in current generation. Although every candidate solution is given a chance to be picked, the candidate with a larger fitness has a higher opportunity to be selected as a parent.

Based on the above selection rule, one pair of parents are selected to create a child by crossover. The task subset of the child is considered as a union of the task subsets of the parents. This novel crossover method guarantees that the child solution is not worse than its parents solutions because the child takes all the tasks of the parents.

After all new children determine their task subsets, the mutation is performed to cover more tasks in their task subsets. Here, we employ a simple mutation method that randomly chooses one task from the entire set of tasks and adds the selected task into the child solution only if this task is not included in its current subset. As a result, the fewer tasks in the subset of the child solution, the higher probability of adding a new task. This mutation method makes a small task subset bigger and keeps a big task subset stable.

The last step is to plan the feasible travelling paths for the new generation based on the tasks subsets after selection, crossover, and mutation. This process is the same as that in the first generation including path planning, feasibility check, and path revision. The final solution is obtained after repeating these operations for a maximum number of generations.

B. DA-based Solution

For problems (6) and (7), the GA is not feasible for two reasons. First, each worker only has access to the information of tasks (e.g., locations, rewards), whereas workers may not be willing to share their own information (e.g., locations, and travelling distance limits) to others. As a result, each worker is not aware of others’ status and intentions. In other words, individual workers do not have full information as the framework does. Second, the GA is an energy and time consuming algorithm, which is not suitable for workers who have limited resources and need to decide their paths within a short time period. Therefore, we propose the detective algorithm (DA) for the worker-centric problem defined in (6) and (7).

As discussed above, the GA takes all tasks into account when planning travelling paths for workers. In the other extreme case, a local search algorithm only considers the nearest task for each individual worker. The GA with the global vision is costly but can approach the optimal solution. In contrast, the near-sighted local search algorithm is cost-effective but restricted in the achievable performance. Here, we consider a compromise of the GA and the local search algorithm in a \( \psi \)-depth detective algorithm (DA). The DA considers \( \psi \) tasks at each step when it plans travelling paths. The larger \( \psi \) is, the more time and energy are needed in evaluating tasks.
Algorithm 2: A DA-based solution for task allocation.

**Input:** $T$ (set of tasks), $W$ (set of workers), $loc_t$ (locations of tasks), $loc_w$ (initial locations of workers), $b$ (budget), $\{l_i | \forall w_i \in W\}$ (maximum travelling distance of workers), $h$ (length of each cycle), $\{f_i | \forall w_i \in W\}$ (moving speed of workers)

**Output:** $\{P_i | \forall w_i \in W\}$ (travelling path of workers)

1. $k \leftarrow 0$ // Number of cycles
2. $total \leftarrow 0$ // Initialize total reward
3. While Maximum travelling distances are not reached do
   // Phase 1: Path planning
4.   $k \leftarrow k + 1$
   //_iteratively plan workers’ paths
5.   for $i$ from 1 to $n$ do
6.     Find the task $t_j$ with the largest profit $g_j$
7.     if $g_j > 0$ then // Take task $t_j$
8.       Add task $t_j$ into set $P_i^k$
9.     else // Detect another task
10.    Assuming task $t_j$ is taken and completed,
11.       find next available task $t_j$ with the largest profit $g_j$ after information update
12.      if $g_j + g_j > 0$ then // Take both tasks
13.         Add task $t_j$ and $t_j$ into set $P_i^k$
14.     // Phase 2: Path selection
15.   End for
16.   while $total < b$ do
17.     Find the path $P_i^k$ with the largest value gain and
18.       the total reward of this path is $r_i^k$
19.     if $r_i^k < (b - total)$ then // Enough budget
20.       Add $P_i^k$ into $P_i$
21.     total $\leftarrow total + r_i^k$
22.     if all paths in cycle $k$ are selected then // Go to next cycle
23.       break
24.   End while
25. return $\{P_i | \forall w_i \in W\}$

Alg. 2 shows an example of the DA-based solution with 2-depth. As seen in Alg. 2, after the initialization (lines 1-2), we assign the tasks to workers cycle by cycle. In each cycle, there are two phases including path planning (lines 5-12) and path selection (lines 13-19). In path planning, DA tries to find a feasible path for every worker. Specifically, it firstly considers the tasks with the highest profit since workers are interested in their profits (line 6). Here, the profit is the surplus of a task reward according to (2) and the cost for a worker to carry out the task according to (1). If the largest profit is larger than 0, the task with the largest profit is added into the path (lines 7-8). Otherwise, assuming that DA has already taken and finished the task with the largest profit (negative profit), it then detects the next available task with the largest profit after information update (lines 9-10). Then DA evaluates these two tasks together to decide whether to take both of them considering the total profit (lines 11-12). Since the cost for workers to finish tasks largely depends on their locations, it is possible that a worker achieves a negative profit if it finishes either of two tasks, while it obtains a positive total profit by finishing both tasks when these two tasks are close in locations. As such, DA has a better chance to take more tasks than the local search algorithm. In path selection, the path candidates generated in the path planning phase are examined and selected to satisfy the budget constraint. Specifically, DA keeps selecting the path with the largest value gain, until the budget runs out or all path candidates generated in current cycle are selected (lines 13-19). Here, the value gain is the increase of total value according to (4) when a new path of a worker is selected. Then, DA moves to the next cycle (line 4) and the task allocation ends when no worker can take over more tasks (line 20).

There are $n$ workers and $m$ tasks. In the worst case, every worker would take all $m$ tasks into its path and only one task is added in each cycle. Then, there are $m$ cycles in the worst case. In the $\psi$-depth DA-based solution, the search of a combination of at most $\psi$ tasks with the largest total profit is $O(n^\psi \psi)$. In the worst case, the largest profit of one search is associated with only one task, which is the task added into the path in current cycle. In phase 1 of path planning, it takes $O(m^\psi)$ to plan the path for one worker in one cycle. There are $n$ workers, so the total time for path planning is $O(nm^\psi)$ in one cycle. In phase 2 of path selection, there are $n$ paths available. In the worst case, all $n$ paths are selected. Since one path is selected when it is the one with the largest value gain among the remaining paths, the time complexity of selecting all $n$ paths is $O(n + (n - 1) + ... + 1) = O(n^2)$. Thus, considering both path planning and path selection in one cycle, the time complexity is $O(nm^\psi + n^2)$. Since there are $m$ cycles in the worst case, the overall time complexity of DA is $O(nm^\psi + n^2)$.

V. Numerical Results and Discussion

In this section, we evaluate the performance of the proposed solutions in terms of the data quality, workers’ profit, and task coverage. We also examine the impact of various parameters on the performance, such as the maximum travelling distance, the initial reward of tasks, the budget, and the length of cycles. In addition, we investigate different geographic distributions of tasks and show the results of performance.

A. The Baseline Solution

As the task allocation problem in (5) is NP-hard, to address the computational intractability, a natural solution is a local search algorithm given in Alg. 3, which is referred to the nearest-first algorithm. It is considered as a baseline solution as it is simple and effective. As seen in Alg. 3, a worker takes the nearest task iteratively until it cannot conduct more tasks. A final solution is achieved after all workers select their feasible travelling paths by the nearest-first algorithm.

The baseline solution is effective but not efficient because it is “short-sighted”. For each worker, the only information is the reward of the nearest task, which prevents the worker from being aware of the task clusters in our scenarios. Workers may be misled to a wrong direction with few tasks or end their travelling paths early when the nearest task has a smaller reward than the cost.
Algorithm 3: A baseline solution for task allocation.

**Input:** \( T \) (set of tasks), \( W \) (set of workers), \( \text{loc}_t \) (locations of tasks), \( \text{loc}_w \) (initial locations of workers), \( b \) (budget), \( \{l_t | \forall w_i \in W\} \) (maximum travelling distance of workers)

**Output:** \( \{P_i | \forall w_i \in W\} \) (travelling path of workers)

```plaintext
for j from 1 to m do
    count[j] ← 0
// Iteratively plan workers’ paths
for i from 1 to n do
    // Initialize path, travelling distance, and initial location of \( w_i \)
    \( P_i \leftarrow \emptyset \)
    distance[i] ← 0
    \( \text{loc}_p[i] \leftarrow \text{loc}_w[i] \)
    Find the nearest task \( t_j \)
    while Budget \( b \) and maximum travelling distance \( l_i \) allow to add task \( t_j \) into \( P_i \) do
        // Update distance, location, and number of data samples
        distance[i] ← distance[i] + ||\( \text{loc}_p[i] - \text{loc}_i[j] ||
        \( \text{loc}_p[i] \leftarrow \text{loc}_i[j] \)
        count[j] ← count[j] + 1
    Find the next nearest task \( t_j \)
    return \( \{P_i | \forall w_i \in W\} \)
```

For the baseline algorithm in Alg. 3, the running time for the loop in lines 1-2 is \( O(m) \). Finding the nearest task in lines 7 and 12 takes time \( O(m) \). The while loop in lines 8-12 runs at most \( m \) times when all \( m \) tasks are added into the path of worker \( w_i \) in the worst case. Thus, the total running time of the while loop is \( O(m^2) \). Therefore, the overall running time of the for loop in lines 3-12 is \( O(nm^2) \), which is also the time complexity of the baseline algorithm. As seen, the time complexity of the baseline solution is lower than that of the GA-based and DA-based solutions.

### B. An Example

To show the difference of these three solutions (baseline, GA-based, and DA-based) for task allocation problem, we provide and analyze a simple example in Fig. 5. In this example, there are three tasks and one worker. For the baseline solution, the worker tries to take the nearest task (task #1), while the profit is negative. Thus, the worker will not take any task and the path found by the baseline solution is empty. For the GA-based solution, the GA aims to achieve a maximum value of data quality (the value is positively related to the number of data samples). As a result, the worker is assigned to take all three tasks. Although the profit of the worker is zero, this assignment is the best solution to the requester-centric problem. In the requester-centric problem, the value of data quality is the top priority. On the other hand, the worker-centric problem focuses on the profit of workers. There is a trade-off between the profit of workers and the value of data quality. As a solution to the worker-centric problem, the DA firstly finds the task with the largest profit. Although task #1 has a negative profit, it is still the task with the largest profit because tasks #2 and #3 are farther to the worker (higher cost) with the lower reward. Then, the DA considers the tasks #1 and #2 together because of the negative profit of task #1. Since the total profit is 1, the worker takes both tasks #1 and #2 into its path. Here, the worker refuses to take task #3 to maximize its profit, which is detrimental to the value of data quality. As seen, the DA sacrifices the value of data quality to improve the profit of the worker from 0 to 1, since the profit of workers is one of the primary objectives in the worker-centric problem.

### C. System Settings

Next, we conduct more comprehensive simulations to evaluate the performance of the task allocation approaches. Here, we set up the sensing region as a 500 × 500 square and generate the positions of tasks and workers therein. There are 50 tasks and 10 workers in the default scenario. The positions of workers follow the uniform distribution. However, the positions of tasks show the clustering effect by applying a cluster method (i.e., the \( k \)-means method) to the uniform distribution. Fig. 2 shows the positions of tasks and workers.

There are several parameters in our formulations of the task allocation problem. In (1) for the cost of taking tasks by each worker \( w_i \), we assume that the cost of each worker has the same parameters: \( \alpha_i = \beta_i = 1, \forall w_i \in W \). For the reward of data samples associated with task \( t_j \) in (2), we assume that all rewards have the same \( \lambda_j = 0.2 \) and \( \gamma_j \) (initial reward) to be considered later. For the value of data quality associated with task \( t_j \) in (4), it is assumed that all tasks have the same \( \delta_j = 2 \) and \( \theta_j = 0.5 \).

On the other hand, our proposed solutions involve several critical parameters. For the GA-based solution, the size of population is set to 1000 and the maximum number of generations is set to 5. For the DA-based solution, the workers are limited by a moving speed \( f_i = 1 \), while the length of cycles \( h \) is discussed later. Table II lists the key parameters and their values.
TABLE II
SIMULATION PARAMETERS.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight of sensing cost $\alpha_i$ in (1)</td>
<td>1</td>
</tr>
<tr>
<td>Weight of travelling cost $\beta_i$ in (1)</td>
<td>1</td>
</tr>
<tr>
<td>Reward reducing rate $\lambda_j$ in (2)</td>
<td>0.2</td>
</tr>
<tr>
<td>Base reward amount $\delta_j$ in (4)</td>
<td>2</td>
</tr>
<tr>
<td>Data value increasing rate $\theta_j$ in (4)</td>
<td>0.5</td>
</tr>
<tr>
<td>Size of population in the GA-based solution</td>
<td>1000</td>
</tr>
<tr>
<td>Maximum generations in the GA-based solution</td>
<td>5</td>
</tr>
<tr>
<td>Moving speed $f_i$ in (6b)</td>
<td>1</td>
</tr>
</tbody>
</table>

D. Effect of Maximum Travelling Distance

The main constraint of workers is the maximum travelling distance, which limits the ranges of workers to explore new tasks. Workers with a larger maximum travelling distance are likely to take more tasks and plan longer paths. To examine the effect of the maximum travelling distance, we assume that all workers have the same limit $l_i$ and vary its value from 100 to 1000 with an interval of 100. Meanwhile, we set the initial reward of tasks ($\gamma_j$) to 100, the budget ($b$) to 50000, and the length of cycles ($h$) to 200.

Fig. 6 shows the values of data quality when the maximum travelling distance varies. As seen, the values of all three solutions are almost the same when the maximum travelling distance is small. Then the values are improved with the increase of the maximum travelling distance, as workers are enabled to take more tasks and plan longer paths. However, the value of the baseline solution keeps unchanged after the maximum travelling distance reaches 500, while the value of the DA-based solution is stabilized after the maximum travelling distance exceeds 800. The reason for the flattening of data values in these two solutions is that, after the saturation points, the maximum travelling distance is no longer the main constraint that limits the data values. In contrast, the trade-off between the cost and reward plays a more important role in the achievable data values.

In Fig. 7, we can clearly see that the average travelling distance is unchanged after these saturation points (i.e., 500 for the baseline solution and 800 for the DA-based solution). As seen, the saturation point of the DA-based solution is much larger than that of the baseline solution. The GA-based solution does not show a tendency of saturation when the maximum travelling distance reaches 1000. This observation in fact demonstrates the different capabilities of these solutions in leveraging the task clustering structure.

The existence of the saturation points is related to the distribution of tasks and the ability of algorithms to leverage task clusters. There will be no saturation point when the algorithm terminates by running out of the maximum travelling distance. That is to say, the saturation point appears when the algorithm cannot make full use of the maximum travelling distance. The baseline solution is not able to leverage task clusters to compensate a temporal loss. As a result, the baseline solution tends to have an early end and a saturation point of a smaller distance limit. The DA-based solution can improve the performance by taking advantage of task clusters to compensate a temporal loss and obtain a longer path. Therefore, the saturation point of the DA-based solution appears later than that of the baseline solution. However, as the DA-based solution restricts the clustering scope, the compensation level is limited. In contrast, the GA-based solution does not limit the task cluster size, and thus further improves the task coverage. It is expected that the GA-based solution would have the largest saturation point among the three solutions if the maximum travelling distance keeps increasing.

Fig. 8 further shows the difference of the three solutions in leveraging task clusters to balance cost and reward. As seen in Fig. 8, the GA-based solution is aware of the task clusters and finds a worker path that goes directly to a task cluster at the very beginning. On the other hand, the baseline solution and the DA-based solution select a different direction to take the nearest task or the task with the largest profit. Then the baseline solution stops after taking two tasks because the third task has a negative profit (too far away). On the contrary, the GA-based solution and the DA-based solution can accept a temporary loss in profit if the subsequent expected reward is able to compensate for the current loss. The compensatory ability of the GA-based solution is better than that of the DA-based solution. As a result, the worker in the DA-based solution ends its path when the total reward of two tasks is not enough for the temporary loss. In contrast, the path in the GA-based solution is much longer due to another task cluster (at the tail of the path).
Fig. 8. Awareness of task clusters.

(a) Path in baseline solution.  
(b) Path in GA-based solution.  
(c) Path in DA-based solution.

Fig. 9. Coverage vs. maximum travelling distance.

Fig. 10. Average profit vs. maximum travelling distance.

Fig. 9 compares the three solutions in terms of task coverage. Here, we consider that a task is covered if it receives at least one data sample. As seen, the GA-based solution has the highest coverage that is nearly 100%, while the DA-based solution is slightly behind. The baseline solution performs the worst in the coverage. This result further verifies the advantage of the GA-based solution in leveraging task clusters and exploring more tasks.

Workers are not interested in the value of data quality but concerned more about the profit. Fig. 10 shows the average profit of workers. We can see that the profits of all three solutions are close when the maximum travelling distance is small, and the profits are improved with the increase of maximum travelling distance. Comparing Fig. 10 for average worker profit with Fig. 6 for total data value, we can clearly observe the trade-off between the value of data quality and the profit of workers. As seen in Fig. 6, the data value of the GA-based solution is greater than that of the DA-based solution. This is a foreseeable result because the targets of the requester-centric problem and the worker-centric problem are not exactly aligned. Workers can improve their profits in the worker-centric task allocation because they have the autonomy to plan their own travelling paths.

Moreover, it is observed in Fig. 10 that, the baseline solution leads to both the smallest data value and the smallest average profit. In addition, the baseline solution and the DA-based solution show similar saturation points with respect to the average profit as to the total data value in Fig. 6. However, the average profit of the GA-based solution decreases after the maximum travelling distance exceeds 700 (almost the same as the saturation point in the DA-based solution). The reason is that the GA-based solution starts to sacrifice the profit of workers for the value of data quality, which has been analyzed in Fig. 5.

In addition to the total value of tasks, the fairness of tasks is another important consideration in task allocation. In our scenario, the fairness of tasks is equivalent to the data balance of tasks. To evaluate the fairness of tasks, we use Jain’s fairness index, which measures whether tasks are receiving a fair share of system resources. Jain’s fairness index is defined as: \( J(n_1, n_2, \ldots, n_m) = \left( \sum_{j=1}^{m} n_j \right)^2 / (m \cdot \sum_{j=1}^{m} n_j^2) \). Here, \( n_j \) is the number of returned data samples of task \( t_j \). The result of Jain’s fairness index ranges from \( \frac{1}{m} \) (worst case) to 1 (best case). The maximum value of 1 is achieved when all tasks receive the same number of data samples. Fig. 11 shows the results of Jain’s fairness index of three solutions with various maximum travelling distances. As seen, the GA-based solution achieves the best fairness under all maximum travelling distances, while the baseline solution is the worst. The fairness indices of the baseline solution and DA-based
solution are stabilized after the saturation points, which are
the same saturation points of total value in Fig. 6. The
fairness indices of the GA-based solution and DA-based solution are
not monotonically increasing with the growth of the maximum
travelling distance, which indicates that the fairness index not
only is influenced by the maximum travelling distance but also
depends on other factors such as the distribution of tasks.

E. Effect of Initial Reward

As an incentive to workers, the reward of tasks influences
the path planning of workers. The cost of a worker for taking
a task mainly depends on its travelling distance and stays the
same if the locations of tasks and workers are determined.
As a result, the profit of finishing a task is directly related to
the task reward. According to (2), the task reward depends on
the initial reward $\gamma_j$ and the decaying rate $\delta_j$, which is set to
0.2 as given in Table II. To examine the effect of the initial
reward, we vary its value from 90 to 190 with an interval of
10. Meanwhile, we set the maximum travelling distance to
1000, the budget to 50000, and the length of cycles to 200.

Fig. 12 shows the average numbers of returned data samples
per task with a varying initial task reward. As seen, the
baseline solution and the DA-based solution obtain more data
samples with the increase of the initial reward, whereas the
GA-based solution is insensitive to the initial reward. This
observation can be easily interpreted as follows. For the
baseline solution and DA-based solution, the task selection
is directly profit-driven. Since the task cost is independent
of the initial reward, a larger initial reward means a higher
profit. Therefore, with the increase of the initial reward, more
tasks can be included in the workers paths. As the DA-
based solution can accept temporary profit loss to explore
more task possibilities, it ends up with a larger task set with
more returned data samples. On the other hand, the GA-based
solution achieves the highest performance in terms of the
average number of data samples, but the result is relatively
stable with the increase of the initial reward. This is because
the GA-based solution has made the best use of the task cluster
structure to optimize the performance and thus leaves little
room for further improvement.

Fig. 13 shows the variation of average profit with the initial
reward. As expected, we can see that the average profits of all
three solutions increase with the growth of the initial reward.
Although Fig. 12 shows that the number of returned data
samples in the GA-based solution is relatively insensitive with
the initial reward, the average profit of the GA-based solution
still improves with the initial reward because of a larger profit
per sample. In contrast, the profits of the baseline solution
and the DA-based solution are improved for two reasons, i.e.,
a larger profit per sample and more returned data samples
(as shown in Fig. 12). As the GA-based solution obtains the
largest number of data samples (also the highest value for
data quality) and the DA-based solution achieves the largest
average profit, we can see the complementary strengths of the
GA-based solution and the DA-based solution. The trade-off
between data value and worker profit still exists even when
the initial reward varies.

F. Effect of Budget

The profit of workers is not only related to the reward
of tasks, but also limited by the budget. In this section, we
conduct the simulations with different budgets changing from
4000 to 18000 with an interval of 2000. We set the maximum
travelling distance to 1000, the initial reward to 100, and the
length of cycles to 200.
Fig. 14 shows the total data values of the three solutions with varying budgets. As seen, in all three solutions, the values of data quality are improved with the increase of budget. The baseline solution has a saturation point at 8000 and the DA-based solution has a larger one at 12000, which means the budget cannot further enable workers to take more tasks beyond these points. This is confirmed in Fig. 15, which shows the total rewards to workers. As seen in Fig. 15, the total reward is stabilized when the budget is sufficiently large.

Fig. 16 shows the average profits of the three solutions with varying budgets. As seen, for the baseline solution and the DA-based solution, the profit is first improved with the increase of budget and then reaches a saturation point. On the other hand, the average profit of the GA-based solution increases constantly with budget. Although the profit of the GA-based solution is the smallest at the beginning, it overtakes the baseline solution when the budget exceeds 14000. However, the final profit of the GA-based solution is still less than that of the DA-based solution.

G. Effect of Cycle Length

In the DA-based solution, we introduce the concept of cycle to solve the competing problem of workers. The smaller the cycle is, the more frequently the reward of tasks is updated. According to the reward mechanism, the task reward for a received data sample decreases with the total number of received data samples for the task. Therefore, with a shorter cycle for more frequent reward updates, this diminishing effect of marginal reward increase is stronger. Correspondingly, this can prevent waste of budget and reduce data redundancy. On the other hand, too frequent updates may lead workers to a blind strolling and give up target tasks frequently. In order to figure out the exact effect of cycle length, we run simulations with the cycle length varying from 100 to 1000 with an interval of 100. We set the maximum travelling distance to 1000, the initial reward of tasks to 100, and the budget to 50000.

Fig. 17 and Fig. 18 show the total data value and the average worker profit with the varying cycle length, respectively. As seen in Fig. 17, the values of the baseline solution and the GA-based solution are not affected by the cycle length since cycles are not implemented therein. The value of the DA-based solution increases with the cycle length, because a larger length means higher rewards of tasks at the beginning of each cycle, which is caused by lagged updates of rewards. Thus, the average reward of each task is improved, which encourages workers to take more tasks. As seen in Fig. 18, the average profit of workers is also improved with a longer cycle length in the DA-based solution because of the improved rewards. Similar to Fig. 17, the average profits of the baseline solution and the GA-based solution stay the same with different cycle lengths. The trade-off between the value and the profit still...
appears with different lengths of cycles.

### H. Distribution of Tasks

The simulations so far are based on the scenario in Fig. 2 with task clusters. In order to investigate the effect of task distributions on performance, we generate several scenarios with different distributions of tasks. First, we generate a scenario with uniformly distributed tasks. Then we apply k-means method to these tasks to create task clusters. The numbers of cluster centers are set to 5, 10, 15, 20, and 25, respectively. The scenarios are shown in Fig. 19 on next page. In the simulations, we set the maximum travelling distance to 1000, the initial reward to 100, the budget to 50000, and the length of cycles to 200.

Table III and Table IV present the performance of the three solutions in terms of the values of data quality and the average profits of workers, respectively. As seen, there is not a consistent variation trend in the value or the profit. However, the trade-off between the data value and the profit is still evident, as GA-based solution achieves the largest value of data quality in all cases and the DA-based solution obtains the highest average profit of workers in most cases (except the case of 25 centers). Obviously, the baseline solution has both the smallest value and the smallest average profit.

It is worth noting that our proposed solutions are not limited to the scenarios with task clusters. They are applicable to various situations. The trade-off between the value of data quality and the average profit of workers is clear in all situations. This conflict is the main motivation for us to address the task allocation problem from two distinct perspectives based on the role of workers. That is, workers can be decision followers or decision makers in MCS according to the targets of task allocation. Requesters or the platform need data samples of tasks as many as possible to guarantee the data quality, while workers are interested in improving their profits. Actually, we cannot achieve two goals simultaneously due to the trade-off between the value of data quality and the profit of workers. Hence, we formulated a requester-centric problem and a worker-centric problem for task allocation to address data redundancy and worker competition separately. The requester-centric problem aims to maximize the value of data quality, while the worker-centric problem considers both the value of data quality and the profits of workers by dividing the sensing activities into cycles. We analyzed the computational hardness of the formulated problems and proved that they are NP-hard. Then, we proposed a GA-based solution and a DA-based solution.

To evaluate the performance of the proposed solutions, we conducted extensive simulations. The results show that the performance of task allocation (i.e., value and profit) is improved with the increase of resources (e.g., maximum travelling distance, initial reward of tasks, budget, and length of cycles). The results also demonstrate that the GA-based solution outperforms the DA-based solution and a baseline solution, as it can leverage task clusters and it can accept the temporary loss for a long-term gain. On the other hand, the DA-based solution provides the highest profits to workers while maintaining reasonable performance for the value of data quality. Last but not the least, the proposed solutions are found efficient in the scenarios with different task distributions.

The GA-based and DA-based solutions improve the performance of task allocation by leveraging the clustering effect in different scales. In the future, it would be interesting to study how to identify task clusters with clustering methods before task allocation. Another promising extension direction is to consider workers with different reputations in task allocation.

### VI. Conclusion and Future Work

In this paper, we investigated the task allocation problem in MCS. In particular, we addressed the clustering effect of sensing tasks since the tasks in practice can be concentrated around popular locations of interest. We considered the task allocation problem from two distinct perspectives based on the role of workers. That is, workers can be decision followers or decision makers in MCS according to the targets of task allocation. Requesters or the platform need data samples of tasks as many as possible to guarantee the data quality, while workers are interested in improving their profits. Actually, we cannot achieve two goals simultaneously due to the trade-off between the value of data quality and the profit of workers. Hence, we formulated a requester-centric problem and a worker-centric problem for task allocation to address data redundancy and worker competition separately. The requester-centric problem aims to maximize the value of data quality, while the worker-centric problem considers both the value of data quality and the profits of workers by dividing the sensing activities into cycles. We analyzed the computational hardness of the formulated problems and proved that they are NP-hard. Then, we proposed a GA-based solution and a DA-based solution.

To evaluate the performance of the proposed solutions, we conducted extensive simulations. The results show that the performance of task allocation (i.e., value and profit) is improved with the increase of resources (e.g., maximum travelling distance, initial reward of tasks, budget, and length of cycles). The results also demonstrate that the GA-based solution outperforms the DA-based solution and a baseline solution, as it can leverage task clusters and it can accept the temporary loss for a long-term gain. On the other hand, the DA-based solution provides the highest profits to workers while maintaining reasonable performance for the value of data quality. Last but not the least, the proposed solutions are found efficient in the scenarios with different task distributions.

The GA-based and DA-based solutions improve the performance of task allocation by leveraging the clustering effect in different scales. In the future, it would be interesting to study how to identify task clusters with clustering methods before task allocation. Another promising extension direction is to consider workers with different reputations in task allocation.
If a worker has contributed sensing data for a sufficiently long time, the quality of its submitted data can be evaluated and the worker with high-quality data thus builds a good reputation. Correspondingly, we can adapt the value function for data samples to take into account the reputations of workers. Then, to maximize the total value of covered tasks, the task allocation solution will allocate more tasks to workers of higher reputations. Such reputation-aware task assignment can improve the crowdsensing performance by selecting more high-quality workers. Nevertheless, the reputation evaluation for workers is challenging and requires long-term data collection and analysis.

REFERENCES


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 is currently a Ph.D. student in the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada. His research interests include Internet of Things and mobile crowdsensing.

Wei Song (M’09–SM’14) received the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2007. In 2009, she joined the Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, where she is now an Associate Professor. Her current research interests include Internet of Things, mobile edge computing, mobile crowdsensing, and device-to-device communications. She received a Best Paper Award from the 2018 IEEE ICC, a 2014 UNB Merit Award, a Best Student Paper Award from the 2013 IEEE CCNC, a Top 10% Award from the 2009 IEEE IM, a Best Paper Award from the 2007 IEEE WCNC. She is the Communications/Computer Chapter Chair of IEEE New Brunswick Section. She co-chaired tracks/symposiums for IEEE VTC Fall 2010, IWCMC 2011, IEEE GLOBECOM 2011, IEEE ICC 2014, IEEE VTC Fall 2016 and IEEE VTC Fall 2017.