Efficient Task Allocation for Mobile Crowd Sensing Based on Evolutionary Computing

Xi Tao and Wei Song
Faculty of Computer Science
University of New Brunswick, Fredericton, Canada
Email: {xtao, wsong}@unb.ca

***Abstract*—Mobile crowd sensing (MCS) offers a promising paradigm for big data collection in a large scale. It leverages the power of mobile smart devices, and shows various advantages over traditional sensing networks, such as high energy efficiency, cost-effectiveness, and flexibility. A key problem in MCS is to efficiently allocate distributed tasks to mobile users (MUs) while addressing various constraints, e.g., in terms of the quality of sensed data and collection cost. In this paper, we further take into account the clustering effect of sensing tasks and propose an efficient approach to solve the NP-hard task allocation problem. In our solution, a variant genetic algorithm (GA) is utilized to maximize the task complete ratio and balance the sensed data among tasks while respecting the MUs' constraints. Considering the unique characteristics of the task allocation problem, we divide the crossover and mutation process of the GA into two steps. First, an available subset of tasks is selected for each MU. Then, a feasible travelling path is designed over this subset of tasks in the second step. The simulation results show that the proposed GA-based solution significantly outperforms the baseline solution in terms of task complete ratio and data balance.**

***Index Terms*—Mobile crowd sensing (MCS), task allocation, genetic algorithm (GA).**

I. Introduction and Related Work

The past decade has witnessed an explosive growth of mobile traffic, pervasive smart devices, and assorted new applications. While this growth leads to increasing revenue to the mobile industry, there are rising concerns on environmental impact and high capital/operating expenditure. Driven by the pressing needs from both economical and ecological perspectives, energy-efficiency and greening becomes one of the dominant themes for the fifth-generation (5G) wireless networks. The resource-rich end devices at the mobile network edge (e.g., smartphones, tablets, and vehicles) can function as fog nodes or cloudlets [1] to facilitate a wide range of applications, such as content distribution [2], information dissemination [3], bandwidth aggregation [4], and environment sensing [5].

In particular, mobile crowd sensing (MCS) offers a promising paradigm for data sensing in a large scale. The smart devices with sensing and computing capacities can collect and upload data to MCS frameworks. Compared to physical sensors deployed in specialized sensing networks, the MCS paradigm is more flexible and cost-effective, and thus attracted great research attentions in recent years. The mainstream studies in MCS are dedicated into two subareas, namely, the MCS applications and the MCS frameworks [6]. In particular, appealing MCS applications include environment monitoring, traffic planning, healthcare, recommendation, and public safety [7]. On the other hand, a MCS framework aims to create a general structure for handling MCS processes including localized analytics, data integrity, and aggregate analytics [8]. According to the involvement of MUs, a MCS framework can involve participatory sensing with active MU engagement and opportunistic sensing without MUs' awareness. From the perspective of delay time, a MCS framework can be considered as deadline-sensitive or delay-tolerant. A deadline-sensitive framework requires that data should be uploaded immediately after sensing, while a delay-tolerant MCS framework allows to separate the actions for sensing and uploading.

In the MCS framework, task allocation is an important issue and the method of task allocation strongly affects energy efficiency and data quality. There have been some studies that design the algorithm for task allocation from different perspectives. As the allocation of sensing tasks among MUs is closely related to participant recruitment, also known as team formation, these two issues are often considered jointly. In [9], Xiao *et al.* solved a deadline-sensitive MU recruitment problem by a greedy algorithm. Guo *et al.* proposed a framework named ActiveCrowd to select MUs for both deadline-sensitive and delay-tolerant tasks based on the movements of MUs [10]. In [11], Xiong *et al.* proposed a framework named EMC^3 to select participants and assign tasks, so that a minimum number of participants can return enough data that cover the target sensing area. It exploits a key idea that piggybacks the upload of sensing data over regular calls to save energy for MUs. In [12], He *et al.* explored the task allocation problem by taking into account location-dependent tasks and adding a geographical limit for MUs via a maximum travelling distance. It proposes a centralized and a distributed near-optimal allocation algorithms. In [13], Cheung *et al.* studied the selection problem for time-sensitive and location-dependent tasks, where users are heterogeneous in their initial locations, movement costs, movement speeds, and reputation levels. Based on a non-cooperative task selection game, an asynchronous and distributed task selection algorithm is developed for each user

This research was supported by Natural Sciences and Engineering Research Council (NSERC) of Canada and NSERC Strategic Network FloodNet.

to compute its task selection and mobility plan that maximizes its total profit (i.e., total reward minus movement cost).

We noticed from the previous works that the locations of sensing tasks and participant users are usually assumed to be randomly distributed in a geographical region [12]. However, this neglects an interesting feature of task distribution in practice that tasks can be clustered around some points of interest. In addition, to guarantee accuracy and reliability of sensed data, a sensing task may require multiple data samples from several independent participants. Meanwhile, the duplicate data samples should be restricted to reduce the unnecessary cost for data redundancy. Hence, a reasonable number of samples can be collected for each sensing task in order to balance the data quality and collection cost. Based on the above insights, in this paper, we explore the effect of clustered sensing tasks on the allocation to MUs. Moreover, we limit the maximum travelling distance of each MU in a sensing cycle such that it is impossible for an MU to take all tasks. As such, we can focus on the more complex scenarios without a trivial solution. In particular, we aim to find a task allocation that maximizes the task complete ratio, i.e., the proportion of tasks that receive the required number of independent samples, while satisfying the MUs' travelling distance limits. To address the computational intractability, we propose a genetic algorithm (GA) based solution for this task allocation problem. The simulation results demonstrate significant performance improvement over a baseline algorithm.

The remainder of this paper is organized as follows. In Section II, we give the system model and the problem formulation. In Section III, we propose a GA-based algorithm to solve the task allocation problem. In Section IV, simulation results are presented to compare the proposed GA-based algorithm with a baseline algorithm. Section V concludes this paper.

II. System Model and Problem Formulation

A. System Model

Consider a MCS scenario that there are a set of sensing tasks and a set of participating MUs, denoted by $T = \{t_1, t_2, ..., t_m\}$ and $U = \{u_1, u_2, ..., u_n\}$, respectively. The locations of the tasks and MUs can be determined via positioning measurements (e.g., GPS). Here, we assume that the initial locations of MUs follow a homogeneous Poisson point process (PPP). In contrast, the tasks are distributed as a cluster point process such as the Matérn process. For a Matérn process, the cluster centers are generated according to another homogeneous PPP, while each cluster contains a random number of children points that are uniformly distributed within a ball of radius r and a center from the preceding PPP. Here, we use the Matérn process to capture the clustering effect, since the sensing tasks can swarm around certain popular places of interest. Nonetheless, it is worth noting that our proposed solution is quite general and not limited to any particular clustering model. Fig. 1 shows an example spatial distribution of sensing tasks and MUs in the MCS scenario.

For each sensing task $t_i \in T$, let η_i denote the minimum number of data samples that are required for t_i. Task t_i is said to be complete when it receives at least η_i data samples that are sensed by η_i distinct participants. This is to ensure independence among the sensed data samples and sufficient data quality since the sensed data may contain possible corruptions and errors. Here, the sensing tasks need to be properly allocated among the participating MUs. As each MU can only devote limited time and energy to the assigned tasks, each MU $u_j \in U$ is subject to a travelling distance limit d_j. Hence, a feasible task allocation to MU u_j must be accompanied by a path P_j that traverses all assigned tasks, and the travelling distance over path P_j, denoted by $d(P_j)$, meets the condition that $d(P_j) \leq d_j$.

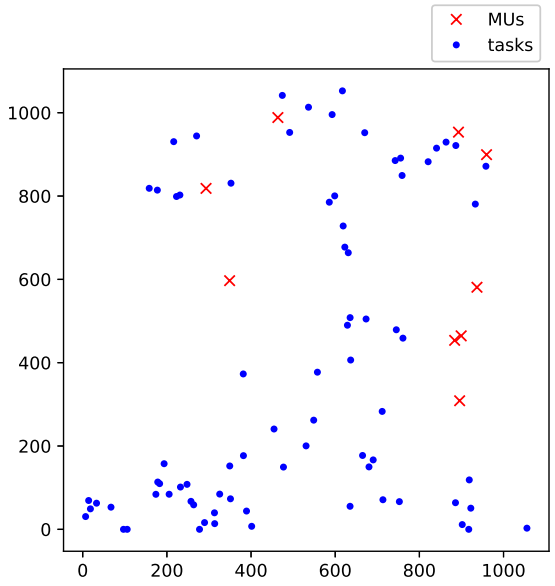

Fig. 1. Example scenario of MCS.

For a subset of assigned tasks and the corresponding traversal path P_j, MU u_j is subject to a total cost $c_j(P_j)$, where the cost function incorporates two primary components as follows:

$$c_j(P_j) = \alpha_j \times |P_{-j}| + \beta_j \times d(P_j). \quad (1)$$

In (1), the first term in the right-hand side represents the sensing cost that is proportional to the number of tasks completed by MU u_j over path P_j. Here, $|P_{-j}|$ indicates the number of vertices along path P_j excluding the first node, which is the initial location of u_j. The second term in the right-hand side of (1) refers to the travelling cost that depends on the travelling distance of the MU. Last, α_j and β_j define the weights for the sensing cost and travelling cost, respectively, in the total cost function.

As data sensing inevitably consumes a variety of resources of the participants, it is essential to reward the participants and compensate for their costs. From the perspective of the sensing task organizer, the goal is to receive at least η_i data samples for each task t_i to ensure sufficient data quality. Any additional data beyond η_i samples will incur unnecessary cost. To suppress the undesired data redundancy, we can vary the rewarding benefit for each received sample according to the

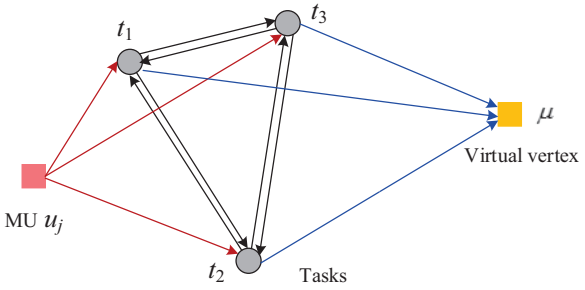

Fig. 2. Construction of the MCP as an instance of the task allocation problem.

total number of samples collected for each task. Specifically, given a task allocation, $\bigcup_{j=1}^{m} P_j$, we use the following function with diminishing marginal increase to determine the rewarding benefit for each sample received for task t_i:

$$b_i(\bigcup_{j=1}^{m} P_j) = \begin{cases} b, & \text{if } \sum_{j=1}^{n} \mathbf{1}(t_i \in P_j) \leq \eta_i \\ b \times e^{\eta_i - \sum_{j=1}^{n} \mathbf{1}(t_i \in P_j)}, & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\mathbf{1}(\cdot)$ is the indicator function and equals 1 when the condition in the function argument is satisfied. Hence, $\sum_{j=1}^{n} \mathbf{1}(t_i \in P_j)$ gives the total number of data samples received for task t_i. As seen in (2), when the number of received data is not more than η_i, the reward is a constant b; otherwise, the reward decreases exponentially with the increase of the number of received data. Hence, when a sufficient number of data have been collected, we can suppress redundant data by reducing the reward gracefully. Thus, given a task assignment P_j for MU u_j, the total benefit that it will receive from the organizer is $\sum_{t_i \in P_j} b_i(\bigcup_{j=1}^{m} P_j)$.

B. Problem Formulation

In order to incentivize participation, we assume that an MU is willing to accept a task assignment only if the cost is not higher than its total rewarding benefit, i.e., $c_j(P_j) \leq \sum_{t_i \in P_j} b_i(\bigcup_{j=1}^{m} P_j)$. Further taking into account the travelling distance limit of each MU, we can formulate the task allocation problem as follows:

$$\max. \quad r \triangleq \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}(\sum_{j=1}^{n} \mathbf{1}(t_i \in P_j) \geq \eta_i) \tag{3a}$$

$$\text{s.t.} \quad d(P_j) \leq d_j, \quad \forall u_j \in U \tag{3b}$$

$$c_j(P_j) \leq \sum_{t_i \in P_j} b_i(\bigcup_{j=1}^{m} P_j), \quad \forall u_j \in U. \tag{3c}$$

Here, the objective function in (3a) is to maximize the task complete ratio, r, i.e., the proportion of tasks that receive at least the minimum number of required data samples. Constraint (3b) defines the travelling distance limit for each MU; and (3c) gives the incentive constraint that each MU's cost cannot exceed the rewarding benefit.

Next, we analyze the computational hardness of problem (3) before introducing our solution.

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

Proof. First, we consider the decision form of problem (3), given by

$$\text{find} \quad P_j, \forall u_j \in U, \text{ with } r \geq \beta \tag{4a}$$

$$\text{s.t.} \quad d(P_j) \leq d_j, \quad \forall u_j \in U \tag{4b}$$

$$c_j(P_j) \leq \sum_{t_i \in P_j} b_i(\bigcup_{j=1}^{m} P_j), \quad \forall u_j \in U. \tag{4c}$$

As seen, the decision form is to decide whether the solution $\{P_j, \forall u_j \in U\}$ achieves an objective value $r \geq \beta$, where β is a given threshold. Here, it takes linear time $O(n)$ to verify constraints (4b) and (4c), and to compare the objective value r with threshold β. Therefore, we prove that problem (4) is NP, since it takes polynomial-time to obtain a yes-or-no answer to the decision problem.

Next, we prove that problem (4) is NP-complete, by reducing a known NP-complete problem, i.e., the multi-constrained path problem (MCP), to an instance of problem (4). The decision form of the MCP is defined as follows. Let $G = (V, E)$ be a graph with weight function $w : E \mapsto Z^+$ and length function $\ell : E \mapsto Z^+$. 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 W and length at most L. It is shown in [14] that the MCP is NP-complete.

In the following, we construct an instance of problem (4) to solve the MCP. Fig. 2 shows an example illustrating the construction. Here, graph G is an *almost* complete graph that connects a single user $u_j \in U$, all tasks $\forall t_i \in T$, and a virtual vertex μ. There is a bidirectional edge between any two vertices, except that there are only unidirectional outgoing edges with the vertex corresponding to user u_j, and unidirectional incoming edges with the virtual vertex μ. Vertex v_1 is mapped to the initial location of user u_j, while vertex v_2 is mapped to vertex μ. Then, for any edge $e \in E$, length ℓ_e is the physical distance between the two ending vertices of edge e, except that length ℓ_e for incoming edges with vertex μ is set to zero. The weight of edge e is defined as

$$w_e = \begin{cases} (\alpha_j + \beta_j \ell_e) - b_i, & \text{if } t_i \neq \mu \\ W, & \text{if } t_i = \mu \end{cases} \tag{5}$$

where task t_i is the tail vertex of edge e. As seen, the weight in (5) basically defines the difference of the cost and the benefit for user u_j to perform task t_i over edge e. Thus, a path that satisfies constraint (4c) is a path with a total weight not more than W. Similarly, constraint (4b) is translated to a path with a total length not more than $L = d_j$, where d_j is the travelling limit of user u_j. Last, the instance of problem (4) has an objective function with threshold $\beta = 0$ for (4a).

Therefore, if there exists a feasible solution to this instance of problem (4), the answer to the MCP is *yes*. Conversely, the answer is *no*. Thus, it is proved that the MCP is reduced to an instance of problem (4). Since the MCP is known to be NP-complete, the decision form of the task allocation problem in (4) is also NP-complete. Hence, the corresponding optimization form defined in (3) is NP-hard.

□

III. The GA-based Solution for Task Allocation

A. The Greedy Solution

As the task allocation problem in (3) is NP-hard, to address the computational intractability, a natural solution is the greedy algorithm given in Alg. 1, which is considered as a baseline solution as it is simple and effective. As seen, Alg. 1 greedily plans a travelling path for each MU one by one. Specifically, an MU selects the nearest task iteratively until the MU cannot take more tasks because the travelling distance limit would be violated or the cost of taking the next nearest task would exceed the benefit. The final solution to the task allocation problem is a set of feasible travelling paths for all MUs.

Algorithm 1 A greedy algorithm for task allocation.

Input: T (set of tasks), U (set of MUs), l_t (locations of tasks), l_u (locations of MUs), $\{\eta_i|\forall t_i \in T\}$ (minimum number of data samples required for tasks), $\{d_j|\forall u_j \in U\}$ (maximum travelling distance of MUs)

Output: P (travelling paths of MUs), r (task complete ratio)

for i = 1 to m **do**
  $count[i] \leftarrow 0$ // Initialize number of received samples
end for
for j = 1 to n **do** // Iteratively assign path for each MU
  $P[:][j] \leftarrow \varnothing$
  $distance[j] \leftarrow 0$ // Initialize travelling distance of u_j
  $l_{sel}[j] \leftarrow l_u[j]$ // Set initial location of path P_j
  find the nearest task t_i
  while it is allowed to add task i into $P[:][j]$ **do**
    $distance[j] \leftarrow distance[j] + \|l_{sel}[j] - l_t[i]\|$
    $l_{sel}[j] \leftarrow l_t[i]$
    $count[i] \leftarrow count[i] + 1$
    find the next nearest task i
  end while
end for
return P and r

As defined in Section II, the tasks appear to be clustered around certain places of interest. In this situation, the greedy algorithm is not an efficient solution because it is "short-sighted", which prevents it from being aware of the clusters of tasks. MUs can be misled to a wrong direction with few tasks. Moreover, another intention of our algorithm is to balance the data collected by the participating MUs among the tasks. If a task receives too much data, the benefit of taking this task will decrease, which jeopardizes the motivation of MUs to complete the task. Unfortunately, the greedy algorithm is unable to address the effect of data redundancy and takes as many tasks as possible.

B. The GA-based Solution

In order to improve the performance, an algorithm with a global vision is needed to optimize the path planning while suppressing data redundancy. In addition, the searching space of the task allocation problem is huge for traditional combinatorial algorithms. Therefore, we propose a variant GA, which can take the clusters of tasks into account during planning the paths for MUs. Meanwhile, the GA-based algorithm can effectively mitigate the data redundancy problem by balancing data collection among the tasks.

Algorithm 2 A genetic algorithm for task allocation.

Input: T (set of tasks), U (set of MUs), l_t (locations of tasks), l_u (locations of MUs), $\{\eta_i|\forall t_i \in T\}$ (minimum number of data samples required for tasks), $\{d_j|\forall u_j \in U\}$ (maximum travelling distance of MUs), pop (size of population), g (maximum number of generations)

Output: P (travelling paths of MUs), r (task complete ratio)

$generation \leftarrow 1$
Initialize task subsets of first generation (pop candidates)
Design travelling paths for the first generation
Check feasibility of all paths and modify any infeasible path to be feasible
Calculate fitnesses of each solution in the first generation
Record the candidate with highest fitness as P and the best fitness as r
while $generation \leq g$ **do**
  $generation \leftarrow generation + 1$
  Apply roulette wheel selection to identify parents for the new generation
  Apply crossover and mutation to create children
  Design travelling paths for the new generation
  Check and make all paths feasible
  Calculate fitnesses for this generation
  Update the best solution P and the best fitness r
end while
return P and r

The GA is an evolutionary computing technique that is inspired by natural selection in biological evolution. Though the GA involves a search procedure that moves toward an optimal solution as in classic heuristic algorithms, the GA applies fundamentally different rules that guide the search procedure. The GA starts from an initial generation with a certain population of individual solutions instead of a single solution. Then, the population evolves iteratively toward an optimal solution. In each iteration, the GA creates a new generation from the current population of solutions based on three main types of rules, namely, *selection*, *crossover*, and *mutation*. For each individual solution in one generation, its *fitness* (usually the objective value of the optimization problem) is evaluated. According to the selection rule, the solutions of higher fitness are 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 solutions to form a child solution, while mutation applies random changes to the genomes that a child inherits from its parents. The GA usually terminates when a maximum number of generations have been produced, or the best fitness has reached a satisfactory level.

For the task allocation problem (3), an individual solution is a set of travelling paths for the selected MUs. As it is hard

to perform the operations of crossover or mutation directly on the travelling paths, we employ a variant GA. In particular, we divide the solution into two steps. The first step selects a subset of tasks for each MU, while the second step determines a travelling path for this subset of tasks. Alg. 2 shows the details of the proposed GA-based solution. Here, we define the fitness as the task complete ratio, which is the objective value of problem (3). The evolution of the GA ends when the number of generations reaches a given threshold.

In the initial generation, we generate a population of individual solutions as follows. First, a subset of tasks are randomly selected for each MU. Then, a travelling path is found by taking the nearest task in the subset until all the tasks in the subset are covered. If the length of the path exceeds the maximum allowed travelling distance, this path is considered as infeasible. To transform an infeasible path into a feasible path, we keep removing the furthest task in the task subset until the travelling distance satisfies the requirement. After all paths of the MUs are turned to be feasible, an individual solution is formed and its fitness is calculated. The first generation contains a number of such individual 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 candidate solution is associated with a probability of selection, which is a proportion of the candidate's fitness over the summation of the fitness of all candidates in the current generation. As a result, every candidate is given a chance to be picked as a parent and the candidate with a larger fitness has a higher opportunity to be selected as a parent.

After one pair of parents are selected according to the above selection rule, crossover and mutation are further applied to create a child for the next generation based on the pair. For the crossover operator, we consider that the task subset of the child is a union of the task subsets of the parents. This crossover method guarantees that the child is not worse than the parents because the child takes all the tasks of the parents. On the other hand, mutation is performed so that more tasks are covered by the child solution. 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 the current subset. As a result, the fewer tasks contained 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 the same. Consequently, data balance is achieved among the tasks. After crossover and mutation, it is possible that there does not exist a feasible path for the child solution. To ensure path feasibility, the child solution is further modified by sequentially removing the furthest task until the path becomes feasible again.

IV. Numerical Results and Discussions

In this section, we evaluate the performance of the proposed GA-based algorithm and compare it with the baseline greedy algorithm in terms of task complete ratio, data balance, and travelling distance.

TABLE I
NUMBERS OF TASKS AND MUS.

Index of Cases	Number of Tasks	Number of MUs
1	78	9
2	68	13
3	105	13
4	91	12
5	72	9
6	87	11
7	149	17
8	131	19
9	103	12
10	90	11

TABLE II
SIMULATION PARAMETERS.

Parameter	Value
Area of sensing region	1000m $\times$ 1000m
Average number of MUs	10
Average number of cluster centers of tasks	20
Average number of tasks in each cluster	5
Range of each cluster	200m
Minimum number of data samples per task	3
Maximum travelling distance of each MU	2000m
Cost coefficient α_j	1
Cost coefficient β_j	1
Basic benefit b	300
Maximum number of generations	5
Size of population	100

A. System Settings

Here, we set up the sensing region as a 1000m $\times$ 1000m square and generate the positions of tasks and MUs therein. The positions of MUs follow a PPP and the average number of MUs is 10. The positions of tasks follow the clustered Matérn process and the average number of cluster centers is 20. In each cluster, it covers a circular region of a radius 200m, and the tasks are distributed within the circle as another PPP with a mean of 5. In the simulations, we generate 10 random cases according to the above settings. The numbers of tasks and MUs for all cases are shown in Table I.

For the GA-based algorithm, we set the maximum generation to 5 and the size of population to 100. The minimum number of data samples required for each task is 3. The maximum travelling distance for MUs is 2000m. In the cost function, the coefficients α_j and β_j are both 1. The basic reward b for the benefit function in (2) is 300. All parameters are listed in Table II.

B. Task Complete Ratio

The most important result of our simulations is the task complete ratio, which is the objective function in our formulation for the task allocation problem. The task complete ratios of GA and the greedy algorithm are shown in Fig. 3. As seen, the GA-based algorithm significantly outperforms the greedy algorithm in terms of the task complete ratio. The improvement percentages of the GA-based algorithm are given in Table III. As seen, the highest improvement is almost 150%, while the minimum improvement is still more than 20%.

TABLE III
COMPARISON OF TASK COMPLETE RATIO.

Greedy Algorithm	GA	Improvement of GA
0.4359	0.7949	82.4%
0.3824	0.9412	146.2%
0.5905	0.7524	27.4%
0.5275	0.7253	37.5%
0.6389	0.8194	28.3%
0.7011	0.9310	32.8%
0.6711	0.8188	22.0%
0.3435	0.8550	148.9%
0.4175	0.6214	48.8%
0.5000	0.7889	57.8%

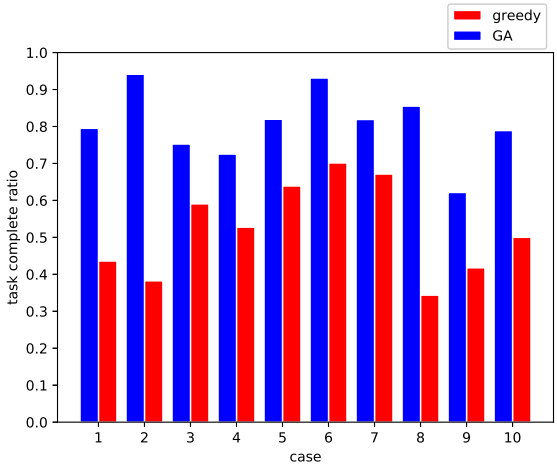

Fig. 3. Result of task complete ratio.

The reason for these improvements can be explained from two different perspectives. First, the GA-based algorithm can plan the paths for the MUs by incorporating more extensive information than the greedy algorithm. To be more specific, the GA-based algorithm can detect the task clusters and make use of such information in the task allocation. Fig. 4 shows an example that illustrates the different solution paths obtained by the two algorithms. As seen, the GA-based algorithm determines a better path for the MU that moves towards the direction of a task cluster. In contrast, the greedy algorithm only makes a local optimal decision at each step and thus cannot leverage the clustering structure in a larger scope. As a result, the greedy algorithm covers fewer tasks within the travelling distance limit.

Second, the GA-based algorithm can accept a temporary loss in the immediate next step as long as the overall benefit is larger than the cost. If a task is too far away from the current location of the MU, the cost to take this task can be so large that it overrides the benefit. Due to the short-sighted nature, the greedy algorithm would give up this task for the MU and end the sensing. On the contrary, the GA-based algorithm can accept such a temporary loss if other tasks are found in the near vicinity to induce more benefits. The example in Fig. 5 demonstrates the different decisions made in the two algorithms. As seen, in the first 14 steps, the MU follows the same path in both algorithms. However, the MU

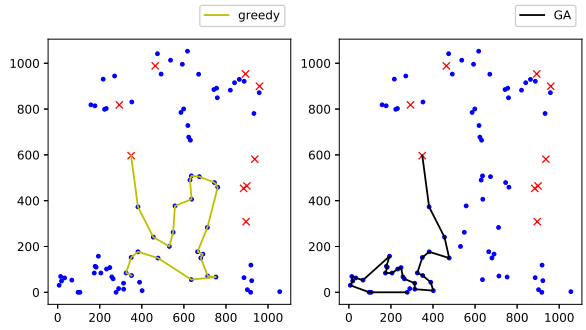

Fig. 4. GA's awareness of the task cluster.

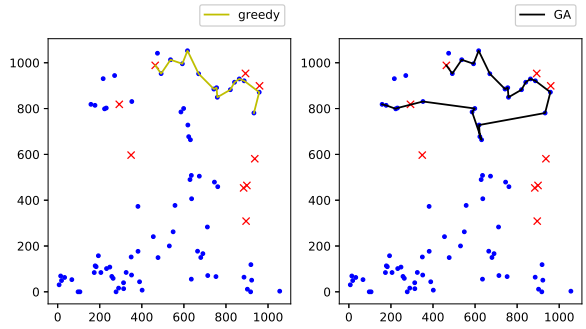

Fig. 5. GA's ability to accept a temporary loss.

in the greedy algorithm terminates after the 14th step, because all remaining tasks are so far away that none can induce a positive profit. As a result, although the travelling distance limit allows a longer path, the greedy algorithm only focuses on the immediate next step and does not explore further. In contrast, the GA-based algorithm locates a better solution via the evolution of generations.

C. Data Balance

In the design of the GA-based algorithm, we also attempt to balance the collected data among the tasks. If the received data are concentrated on a small number of tasks, it would result in a low task complete ratio since many tasks may end up with insufficient data. In addition, the redundancy of the received data for a task decreases the rewarding benefit for taking it, which further jeopardizes the willingness of the MUs to finish the task. Therefore, we design the mutation operator for the GA-based algorithm in a way that balances the collected data among the tasks. Fig. 6 shows the proportion of the tasks that receive *exactly* 3 data samples (the required minimum number) over the total number of tasks. As seen, the GA-based algorithm achieves a larger proportion over the greedy algorithm for all 10 cases. This observation demonstrates the effectiveness of the GA-based algorithm in data balance.

D. Travelling Distance

As mentioned in Section II, any MU cannot take all the tasks without violating the travelling distance limit. Hence, the travelling distance is an important metric to evaluate the performance of the task allocation solution. Fig. 7 shows the

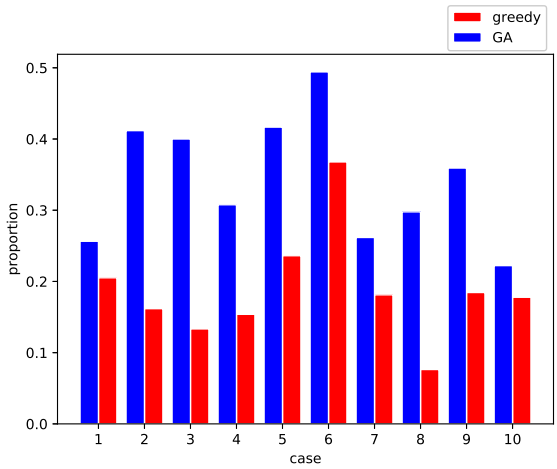

Fig. 6. Result of data balance.

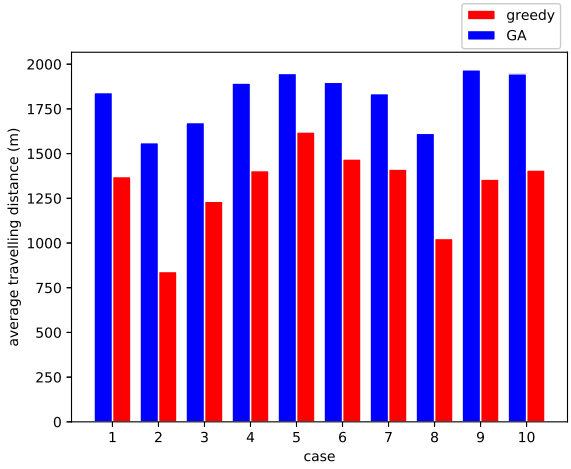

Fig. 7. Result of travelling distance.

average travelling distance of all MUs in all 10 cases. It can be seen that the average travelling distance of the GA-based algorithm is longer than that of the greedy algorithm in every case. As shown in Section IV-B, the GA-based algorithm can accept a temporary loss as long as a longer path is more profitable. Due to the inherent clustering structure of tasks, it is likely that there exists a task cluster that can easily make up for the current loss. Although the GA-based algorithm results in solutions of longer paths, all paths satisfy the maximum travelling distance limit. In other words, the GA-based algorithm exploits the distance limit to the most in order to complete more tasks.

Last, it is worth noting that the GA-based algorithm takes more time to search the solution space although it performs better than the greedy algorithm. Nonetheless, we can reduce the computing time by choosing a smaller population and fewer generations if a lower complete ratio is acceptable.

V. Concluding Remarks

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 formulated the task allocation problem by taking into account realistic factors, such as the MU's travelling distance limit and incentive constraint that the rewarding benefit to an MU cannot be lower than its collection cost. The formulated problem aims to maximize the task complete ratio, which is the proportion of the tasks that receive a sufficient number of independent data samples from the MUs to ensure the sensing quality. As the task allocation problem is NP-hard, we proposed a GA-based algorithm to derive the travelling paths for the participating MUs. The proposed solution also attempts to achieve a good data balance among the sensing tasks. This is because unbalanced data not only degrades the complete ratio but also results in undesired data redundancy that jeopardizes the willingness of MUs to finish the tasks. The simulation results show that the GA-based algorithm outperforms the baseline algorithm in terms of task complete ratio and data balance. The performance improvements can be attributed to the advantages of the GA-based algorithm in leveraging the task clustering structure and accepting temporary loss for long-term gain.

References

[1] X. Wang, Y. Sui, J. Wang, C. Yuen, and W. Wu, "A distributed truthful auction mechanism for task allocation in mobile cloud computing," *IEEE Trans. Services Comput.*, 2018, to appear.

[2] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. on Mobile Comp.*, 2017, to appear.

[3] W. Song, Y. Zhao, and W. Zhuang, "Stable device pairing for collaborative data dissemination with device-to-device communications," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1251–1264, 2018.

[4] D. Zhou, W. Song, P. Wang, and W. Zhuang, "Multipath TCP for user cooperation in LTE networks," *IEEE Network*, vol. 29, no. 1, pp. 18–24, 2015.

[5] J. Wang, J. Tang, X. Sheng, G. Xue, and D. Yang, "Enabling green mobile crowd sensing via optimized task scheduling on smartphones," in *Proc. IEEE Int. Conf. Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.

[6] D. Zhang, L. Wang, H. Xiong, and B. Guo, "4W1H in mobile crowd sensing," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 42–48, 2014.

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

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

[9] M. Xiao, J. Wu, H. Huang, L. Huang, and C. Hu, "Deadline-sensitive user recruitment for mobile crowdsensing with probabilistic collaboration," in *Proc. IEEE 24th Int. Conf. Network Protocols (ICNP)*, 2016, pp. 1–10.

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

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

[12] S. He, D.-H. Shin, J. Zhang, and J. Chen, "Near-optimal allocation algorithms for location-dependent tasks in crowdsensing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3392–3405, 2017.

[13] M. H. Cheung, R. Southwell, F. Hou, and J. Huang, "Distributed time-sensitive task selection in mobile crowdsensing," in *Proc. ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2015, pp. 157–166.

[14] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, pp. 95–116, 1984.