Multi-Task Allocation Under Time Constraints in Mobile Crowdsensing

Xin Li and Xinglin Zhang, *Member, IEEE*

Abstract—Mobile crowdsensing (MCS) is a popular paradigm to collect sensed data for numerous sensing applications. With the increment of tasks and workers in MCS, it has become indispensable to design efficient task allocation schemes to achieve high performance for MCS applications. Many existing works on task allocation focus on single-task allocation, which is inefficient in many MCS scenarios where workers are able to undertake multiple tasks. On the other hand, many tasks are time-limited, while the available time of workers is also limited. Therefore, time validity is essential for both tasks and workers. To accommodate these challenges, this paper proposes a multi-task allocation problem with time constraints, which investigates the impact of time constraints to multi-task allocation and aims to maximize the utility of the MCS platform. We first prove that this problem is NP-complete. Then two evolutionary algorithms are designed to solve this problem. Finally, we conduct the experiments based on synthetic and real-world datasets under different experiment settings. The results verify that the proposed algorithms achieve more competitive and stable performance compared with baseline algorithms.

Index Terms—Mobile crowdsensing, multi-task allocation, time constraint, evolutionary algorithm

1 Introduction

In recent years, mobile crowdsensing (MCS) [1] is becoming a popular sensing paradigm to take advantage of the collective sensing capabilities of the large population of mobile users. Unlike traditional sensing networks which rely on dedicated sensor deployment, any user carrying a mobile device (such as smartphones and tablets) equipped with multi-functional sensors can become a data source in MCS. For example, users can report the road-surface condition on the way home for traffic surveillance [2], [3], or collect noise information for noise pollution monitoring while walking after a meal [4]. At the same time, a user can also play the role of information requester by actively asking some mobile users to collect sensed data to fulfill their requirements. Thanks to its efficiency and scalability in collecting various types of sensed data, MCS has a wide range of applications, such as public safety [5], [6], environment monitoring [6], [7], health care [8], signal map construction [9] and indoor localization [10].

In general, there are three roles in MCS applications: workers (i.e., users who undertake sensing tasks), requesters (i.e., users who send task requests), and the MCS platform. The MCS platform is responsible for allocating sensing tasks to suitable workers and integrating sensed data for the corresponding task requesters. Considering that the numbers of workers and tasks can be rather large, a proper task allocation scheme is important to match workers with tasks, such that the MCS applications can operate efficiently and improve user stickiness.

• *The authors are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510641, China. E-mail: lixin_forget@163.com, zhxlinse@gmail.com.*

Manuscript received 2 June 2018; revised 12 Nov. 2019; accepted 22 Dec. 2019. Date of publication 30 Dec. 2019; date of current version 4 Mar. 2021. (Corresponding author: Xinglin Zhang.) Digital Object Identifier no. 10.1109/TMC.2019.2962457

As an important part of four-stage life cycle (i.e., task creation, task assignment, individual task execution, and crowd data integration) in mobile crowdsensing process [11], task allocation has become a crucial research issue in MCS and drawn a lot of research attention [12], [13], [14], [15]. These works mostly consider the single-task allocation scenario, where an available worker is associated with one task at one round of task allocation. In this paradigm, if a worker is willing to undertake multiple tasks for rewards, he has to wait and interact with the MCS platform for multiple rounds of assignments. Furthermore, considering that most MCS tasks are location dependent, workers are required to physically travel to the locations of interest in order to complete the allocated tasks. In the single-task allocation scenario, a worker may be assigned with tasks that have large accumulated traveling distances when undertaking multiple tasks during his valid working period.

Considering these inefficient factors, recent research efforts try to model multi-task allocation frameworks [16], [17], [18], [19], [20], [21]. Some of these works take into account the valid time of a task, as many sensing tasks are time sensitive (such as traffic dynamic monitoring and pollution monitoring at specified locations and time intervals). However, they do not incorporate the worker's time availability. Intuitively, a mobile worker can spend limited time to perform sensing tasks each day. Some researchers hence start to consider time availability of both tasks and workers [21], but they only consider the scenario where a worker has a destination, which means that the tasks that can be completed should be located along the worker's moving direction. This model limits the sensing capability of mobile workers who do not have explicit destinations. For example, a worker may want to perform sensing tasks within half an hour of his after-meal walking time in the neighborhoods, or a worker is in the shopping

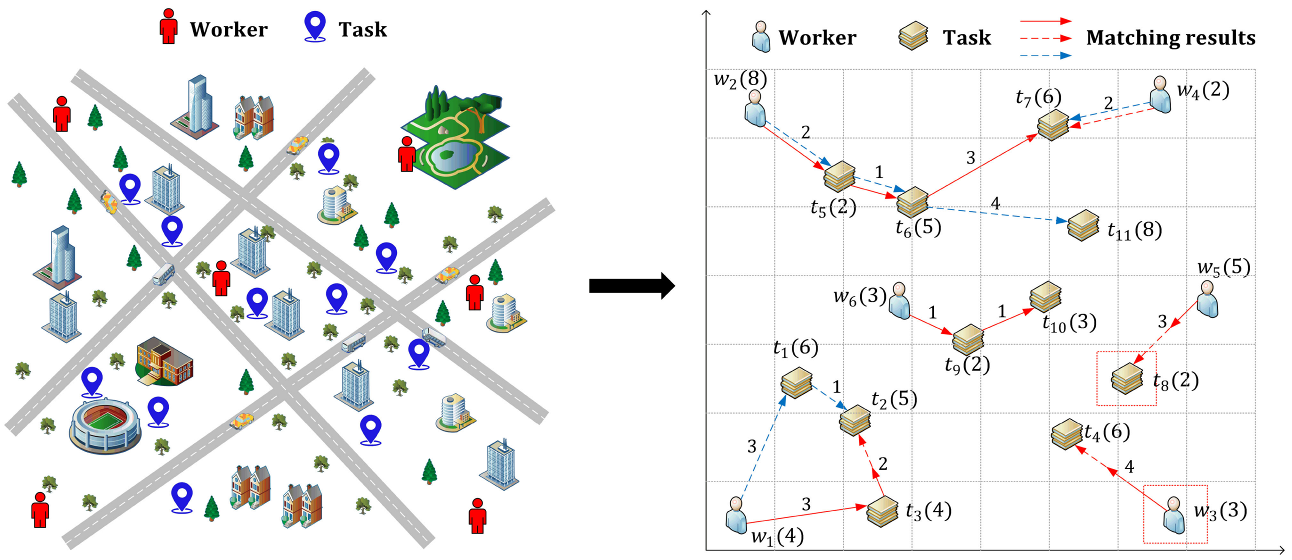


Fig. 1. An example of task allocation. The left figure shows the distribution of some workers and tasks in a certain sensing area, and the right figure is a distribution map of tasks and workers mapped according to the left figure. As shown in the right figure, the line segments indicate the candidate matching results between workers and tasks, where the red solid lines represent a set of feasible solutions, the red dotted lines indicate allocation failure (infeasible solutions), and the blue dotted lines indicate a set of better solutions. The numbers on the line segments represent the time costs. For example, it takes 3 time units from the location of w_1 to the location of t_3. The number in parentheses indicates the valid time of tasks or working time of workers. For example, $w_2(8)$ means worker w_2 has 8 available time units, while $t_2(5)$ means task t_2 will expire after 5 time units.

zone where he can spend a certain amount of time wandering and thus can perform sensing tasks. By performing the assigned tasks, these workers can receive some rewards. Furthermore, similar to the traditional crowdsourcing platform MTurk [22] where there are a lot of workers who complete crowdsourced tasks to earn money when they have time, we can envision that there will be an MCS platform where there are many mobile workers who want to earn rewards by completing sensing tasks when they have some time. Therefore, in this work, we investigate efficient task allocation schemes in this general multi-task allocation scenario where both tasks and workers are associated with valid time constraints. Specifically, workers should only be assigned with the tasks in the positions where the workers can arrive during their available time; on the other hand, tasks should be assigned to workers who can arrive at the target location within the effective time of the tasks.

Take Fig. 1 as an illustrating example. The workers and tasks are associated with valid time units. The line segments indicate the candidate matching results between workers and tasks. The time units it takes to travel between the locations of workers and tasks are listed beside the line segments. Considering the time constraints, the assignment of worker w_6 to tasks t_9 and t_{10} is a feasible solution, while the assignments of w_5 to t_8 and w_3 to t_4 are infeasible due to the time constraint violation. Several challenges occur when designing efficient multi-task allocation approaches with time constraints:

- Consider worker w_1 with 4 available time units. There are three tasks located around w_1, i.e., t_1, t_2, t_3 with valid time units of $6, 5, 4$, respectively. As can be seen, if we assign t_3 to w_1 first by the commonly adopted greedy strategy, w_1 can be assigned with one task as the next closet task t_2 will be expired when w_1 moves from t_3 to t_2. On the hand, we may assign t_1 to w_1 at first according to the same strategy. In this case, t_1 and t_2 can be successfully be assigned to w_1. This phenomenon indicates that, given the time constraints of workers and tasks, it becomes more difficult in determining the most suitable bunch of tasks for a single user.
- Consider the competition of two workers w_2 and w_4. If we assign tasks to fulfill w_2, an efficient and feasible task sequence is t_5, t_6, t_7 according to the traveling time. However, in this case, w_4 can not be assigned with any task since the only accessible task to w_4 is t_7, which has been occupied. On the other hand, if w_2 is assigned with the feasible task sequence t_5, t_6, t_{11}, w_4 can be mapped to t_7. From this aspect, the overall utility of the MCS is improved. In summary, given time constraints of workers and tasks, the competition among workers and tasks become more complex when designing efficient task allocation methods.

Considering these challenging scenarios, we propose efficient multi-task allocation schemes with time constraints based on genetic algorithm (GA) [23], which has been proved to be efficient in solving complex combination optimization problems as illustrated above. In brief, we have made the following contributions in this paper:

1) We formulate a general multi-task allocation problem with time constraints for both workers and tasks, and prove that the formulated problem is NP-complete.
2) We propose two heuristic algorithms, namely MATC-GA and MATC-IGA, to efficiently solve the formulated task allocation problem.
3) We evaluate the proposed algorithms based on synthetic and real-world datasets. The experimental results show the superiority of our algorithms compared with other algorithms.

The rest of the paper is organized as follows. Section 2 summarizes the related works. Section 3 analyzes and

TABLE 1
Main Notations

Notation	Description
W, w_j	the worker set and a worker
wl_j, wt_j	the location and expected working time of w_j
T_{w_j}, wa_j	the task set to be allocated to w_j and the size of the task set
T, t_i	the task set and a task
tl_i, te_i	the location and valid time of task t_i
ta_i, wr_i	the utility obtained by the platform and the reward obtained by the worker after task t_i is allocated
te_c	current time

demonstrates the multi-task allocation problem with time constraints. Section 4 proposes two heuristic algorithms in detail to solve the studied problem. In Section 5, we evaluate the proposed algorithms and discuss the results. Finally, the discussions and conclusions are drawn in Sections 6 and 7.

2 Related Work

With the development of MCS applications [24], [25], [26], [27], [28] and platforms [29], [30], [31], task allocation becomes a key issue that influences the efficiency of MCS. Many exiting works on task allocation considers the single-task allocation scenario. Zhang *et al.* [12] propose a framework, named CrowdRecruiter, to select workers to satisfy the probabilistic coverage constraint under the piggyback crowdsensing paradigm while minimizing incentive payments. In the similar sensing setting, Xiong *et al.* [13] solve a bi-objective task allocation problem with the aim of maximizing the probabilistic coverage under a fixed budget. Reddy *et al.* [14] study a recruitment framework that selects appropriate workers for data collection to maximize spatial coverage. Considering the task allocation problem for heterogeneous workers with different initial positions, moving costs, moving speeds and reputation levels, Cheung *et al.* [15] propose an asynchronous distributed task allocation algorithm. Cheng *et al.* [32] model a task allocation model by considering workers' observation angles and time when they undertake tasks, and propose an effective algorithm to solve this problem. Zhang *et al.* [58] formulate a reliable task assignment problem which incorporates the quality of sensing tasks and propose effective algorithms with theoretical guarantees.

Recently, researchers have begun to study the multi-task allocation scenario, where the potentials of workers can be made full use of Zhang *et al.* [16] propose a bi-objective crowdsensing model, aiming at recruiting a set of vehicles to simultaneously complete location-based query tasks and automatic sensing tasks while maximizing the sensing utility of each participant. The task allocation optimization model proposed by He *et al.* [17] aims to maximize the rewards for the MCS platform, given that each worker has a moving distance budget and each location-based task needs to be completed by multiple workers. Song *et al.* [18] study the multi-task allocation strategy to minimize the number of selected participants under the total budget constraints and the requirements of quality information for concurrent tasks. Considering the different requirements of time-sensitive tasks and delay-tolerant tasks, the framework ActiveCrowd proposed by Guo *et al.* [19] study two multi-task allocation situations: task allocation based on workers' intentional movement for time-sensitive tasks and unintentional movement for delay-tolerant tasks. Liu *et al.* [20] study two scenarios for multi-task allocation optimization: 1) maximizing the number of completed tasks while minimizing the total traveling distance, and 2) minimizing the incentive cost while minimizing the total moving distance. In addition, Li *et al.* [33] also study multiple heterogeneous task allocation, and propose a dynamic recruitment algorithm aiming to minimize the incentive cost.

Moreover, there are a few studies that consider the impact of time constraints on task allocation. Cheung *et al.* [15] propose a model that collect time-sensitive and location-dependent information by heterogeneous workers to maximize coverage. Guo *et al.* [19] take into account the impact of tasks with different time types on task allocation, and propose two greedy-enhanced genetic algorithms to address them. In addition, Deng *et al.* [34] propose a worker selected tasks mode, and consider a set of tasks with locations and expiration time. The goal is to find a schedule for workers that maximizes the number of performed tasks.Estrada *et al.* [35] propose a service computing framework for time constrained-task allocation in location-based crowdsensing systems. The platform goal is to efficiently determine the most appropriate set of workers for each task so that high-quality results are returned within the requested response time.

However, few studies take into account the time constraints of both workers and tasks in a multi-task scenario. In this work, we investigate this scenario and formulate a general task allocation model for MCS, and we design two efficient heuristic algorithms for solving the task allocation problem. Note that Zhao *et al.* [21] also consider the time constraints of workers. But they study a destination-aware task assignment problem to maximize the total number of completed tasks while all workers can reach their destinations before deadlines and tasks assigned to workers can be completed before expiration. Their problem can be considered as a special scenario of our proposed model to some extent.

Note that the specific structural properties (e.g., optimization goals, spatial-temporal constraints, coverage constraints) of task allocation problems in MCS make the solutions from other domains difficult to be directly applied in MCS. For example, task (or resource) allocation schemes in edge computing [36] and cloud computing [37] do not consider the spatial-temporal constraints as proposed in this paper, hence their solutions cannot be applied directly. Due to the page limit of the paper, we have made a table describing the different concerns in the relevant domains of MCS [38], [39], edge/cloud computing [37], [40], [41], robot/UAV [42], [43] in Appendix A of the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2019.2962457.

3 Problem Statement

In this section, we present the main definitions and the formulated problem. For clarity, the main notations are summarized in Table 1.

Definition 1 (Worker). *A worker w_j is willing to undertake multiple sensing tasks, subject to a constraint that the total traveling time starting from his current location wl_j is not larger than his expected working time wt_j.*

We assume that the workers are ordinary mobile users who are motivated to perform tasks for the following factors: 1) Workers have vacant time to perform sensing tasks by moving around, thus they do not have explicit requirement on traveling cost as long as they can arrive at the locations of tasks within the time budget; 2) Workers can receive rewards after completing sensing tasks. We also assume that the travel time from one place to other place is proportional to the movement distance (such as [19], [20]) and that each worker has sufficient battery power to perform tasks.

Definition 2 (Task). *A sensing task t_i submitted by a requester is associated with a location tl_i, a revenue tr_i and a valid period $[te_c, te_c + te_i]$, where te_c is the current time and te_i is the time duration.*

We assume that the sensing tasks are easy for human by using the sensing devices (e.g., smartphones), and the sensing tasks are heterogeneous in the sense that they have different spatial and temporal constraints and require different sensing dimensions and skills (such as taking photos and recording noise at different time and locations). In our scenario, the tasks do not have explicit priorities. However, the task revenue posted by the requester can reflect the priority to some extent in the formulated optimization problem.

Definition 3 (MCS platform). *An MCS platform is responsible for allocating sensing tasks to suitable workers and integrating sensed data for the corresponding task requesters. If a task t_i is assigned to a worker w_j, the MCS platform will receive the revenue tr_i of t_i given by the requester, while the worker will get a reward wr_i given by the MCS platform. Therefore, the utility of the platform can be calculated as $ta_i = tr_i - wr_i$ after the task t_i is assigned.*

Given the above definitions, once a worker wants to undertake sensing tasks, he submits his information which includes his current location and expected working time to the MCS platform. Similarly, a requester can submit a sensing task with required information, such as its location, revenue and valid period. Note that the worker reward determination method adopted by the MCS platform can vary according to specific applications (e.g., setting the reward to be a fixed equal value, or be in direct proportion to the revenue of the task). The metric of the platform utility or worker reward depends on the "currency" adopted in the MCS application. Common metrics could be dollars or electronic cashes.

Consider that there is a worker set $W = \{w_1, w_2, \ldots, w_m\}$ (where m is the number of workers) and a task set $T = \{t_1, t_2, \ldots, t_n\}$ (where n is the number of tasks). We define a traveling distance to assist in verifying whether the constraints of workers and tasks are met respectively, i.e., whether the workers can complete the assigned tasks within the expected working time and whether the tasks can be completed within the valid period.

Definition 4 (Traveling distance). *Given that the set of ordered tasks allocated to w_j is $T_{w_j} = \{t_1, t_2, \ldots, t_{s_j}\} \subseteq T$, where the task t_{s_j} is the last task, we define the function $f_D(w_j, t_i, T_{w_j})$ to represent the traveling distance from the initial location wl_j of worker w_j to the location of task $t_i \in T_{w_j}$. The function $f_D(w_j, t_i, T_{w_j})$ can be expressed as follows:*

$$f_D(w_j, t_i, T_{w_j}) = \begin{cases} f_d(wl_j, tl_1) & wa_j = 1 \\ f_d(wl_j, tl_1) + \sum_{i'=2}^{i} f_d(tl_{i'-1}, tl_{i'}) & wa_j \geq 2 \end{cases}, \tag{1}$$

where wa_j is the size of the task set T_{w_j} and the distance from location a (task or worker) to location b (task or worker) is denoted as $f_d(a, b)$.

The goal of the MCS platform is to maximize its utility by matching workers to the suitable tasks. Then the studied problem of multi-task allocation with time constraints (MATC) can be defined as follows:

Definition 5 (MATC). *The problem of MATC is to solve the following optimization problem:*

$$\max\left(\sum_{j=1}^{m}\sum_{i=1}^{n} ta_i \cdot x_{ji}\right), \tag{2}$$

subject to

$$f_D(w_j, t_{s_j}, T_{w_j}) \leq wv \cdot wt_j, \quad \forall w_j \in W \tag{3}$$

$$f_D(w_j, t_i, T_{w_j}) \leq wv \cdot te_i, \quad \forall t_i \in T_{w_j} \tag{4}$$

$$x_{ji} = 1 \text{ or } x_{ji} = 0, \quad \forall w_j \in W, t_i \in T \tag{5}$$

$$\sum_{j=1}^{m} x_{ji} \leq 1, \quad \forall t_i \in T, \tag{6}$$

where wv is the traveling velocity, x_{ji} indicates the status of task assignment: $x_{ji} = 1$ indicates that task t_i has been assigned to worker w_j, while $x_{ji} = 0$ means that task t_i has not been assigned to worker w_j.

According to Definition 5, MATC is a combinatorial optimization problem which maximizes the utility of the MCS platform. Eqs. (3) and (4) demonstrate the constraints of workers and tasks respectively, i.e., the workers can complete the assigned tasks within the expected working time and the tasks can be completed within the valid period. Eqs. (5) and (6) give the restriction on tasks that the status of each task is assigned or unassigned and each task can only be assigned to at most one worker respectively.

The solution space of MATC is very large and finding the optimal solution is difficult. Assuming that there are n tasks and m workers, and a worker can complete more than one task while a task can only be allocated to one worker. There are $m + 1$ ways to allocate a task. For all of n tasks, there will be $(m + 1)^n$ allocation ways. In addition, the order in which tasks are completed will greatly affect the allocation. In summary, the solution space will reach $(m + 1)^n \cdot n!$. Therefore, it is impractical to enumerate all possible allocations and it is difficult to find a reasonable solution in a reasonable time. In fact, we can prove that the MATC problem is NP-complete.

Lemma 1. *The MATC problem is NP-complete.*

Proof. The Multiple Knapsack Problem (MKP) has been proved to be an NP-complete problem [44]. MKP can be described as follows: Given a set $U = \{u_1, u_2, \ldots, u_n\}$ of n items and a set $V = \{v_1, v_2, \ldots, v_m\}$ of m knapsacks, where $m \leq n$. For $u_{i'} \in U$, its weight is defined as $s_{i'}$ and

TABLE 2
The Mapping Between MKP and Simplified MATC

MKP	SIMPLIFIED MATC
$U = \{u_1, u_2, \ldots, u_n\}$	$T = \{t_1, t_2, \ldots, t_n\}$
$V = \{v_1, v_2, \ldots, v_m\}$	$W = \{w_1, w_2, \ldots, w_m\}$
$v_{j'} \in V, c_{j'}$	$w_j \in W, wt_j$
$u_{i'} \in U, s_{i'}, p_{i'}$	$t_i \in T, te_i + \mathcal{D}, ta_i$
$\max(\sum_{j'=1}^{m} \sum_{i'=1}^{n} p_{i'} x_{j'i'})$	$\max(\sum_{j=1}^{m} \sum_{i=1}^{n} ta_i x_{ji})$
$x_{j'i'} = 0$ or 1	$x_{ji} = 0$ or 1

its utility is defined as $p_{i'}$. The capacity of $v_{j'} \in V$ is defined as $c_{j'}$. The objective is to find a viable way to put items into knapsacks, so that the total utility of all items in the knapsacks is maximized. The MKP is to solve the optimization problem: $\max(\sum_{j'=1}^{m} \sum_{i'=1}^{n} p_{i'} x_{j'i'})$, where the $x_{j'i'}$ indicates the status of items. $x_{j'i'} = 1$ indicates that item $u_{i'}$ has been put into knapsack $v_{j'}$, while $x_{j'i'} = 0$ means that item $u_{i'}$ has not been put into knapsack $v_{j'}$.

We prove the lemma by reducing the MKP to an instance of MATC problem. First, we assume that the time a worker spends to move from one location to another is a fixed value $\mathcal{D}$. x_{ji} indicates the status of tasks. $x_{ji} = 1$ indicates that task t_i has been allocated to worker w_j, while $x_{ji} = 0$ means that task t_i has not been allocated to worker w_j. The simplified MATC is to solve the optimization problem: $\max(\sum_{j=1}^{m} \sum_{i=1}^{n} ta_i x_{ji})$. Next we can associate MKP and simplified MATC by mapping U to T, V to W, $\{v_{j'}, c_{j'}\}$ to $\{w_j, wt_j\}$, $\{u_{i'}, s_{i'}, p_{i'}\}$ to $\{t_i, (te_i + \mathcal{D}), ta_i\}$, and $x_{j'i'}$ to x_{ji}, as shown in Table 2.

As demonstrated above, the MKP is as complex as the simplified MATC, which means that the simplified MATC is also NP-complete, which completes the proof. □

4 Task Allocation Algorithms for MATC

From the above section, we can see that the solution space of the MATC problem is too large for traditional combinatorial optimization algorithms. When the scale of the problem increases gradually, the exact algorithms cannot return results in polynomial time. Therefore, we consider designing efficient heuristic algorithms to solve the problem. Specifically, considering that the characteristics of Genetic Algorithm are suitable for the proposed MATC problem, we will design task allocation schemes based on GA. In this section, we first design a genetic algorithm for MATC according to the problems and constraints. In order to further improve the performance of the algorithm, we then design the immune genetic algorithm based on vaccine for MATC. Since effective constraint handling techniques for evolutionary strategies are important for finding superior solutions [45], [46], we customize several constraint handling strategies in the key operators (including the selection operator, repair operator, and IGA_selection operator) in our algorithms.

4.1 Genetic Algorithm for MATC

A traditional GA generally contains chromosome representation, fitness evaluation, selection operator, crossover operator and mutation operator. We design a new *genetic algorithm for MATC* named *MATC-GA* based on the idea of GA and the characteristics of MATC. In GA, it is important to construct a suitable chromosome which can not only express the content of the problem, but also reduce the computational complexity. In existing works, GAs mainly adopt the traditional 0-1 encoding structure (such as [19]) or the fixed-length real number encoding structure (such as [16]) as the representation of the solution. However, in MATC, these schemes cannot efficiently represent the solution structure and will result in more complicated subsequent operations. Therefore, we design an appropriate encoding structure for the solution of MATC. Based on the constructed chromosome, the proposed MATC-GA is sketched in Algorithm 1. Line 1 generates initial population with a specified size N. Then, the main loop (Lines 4-11) iteratively selects better individuals to find the best solutions. In Line 5, we compute the fitness values of all chromosomes based on function $f(\cdot)$, and update the currently best chromosome which has the best fitness (line 6). Line 7-9 perform selection, crossover, and mutation operations in sequence. The loop ends when the *Maximum Generation* set in advance is met. We next explain the key components in Algorithm 1.

Algorithm 1. MATC-GA

```
Input: Task set $T$, Worker set $W$
Output: Best chromosome (task-worker allocation scheme)
1: Initialize population $G(0)$ with $N$ chromosomes
2: $P_0 \leftarrow G(0)$
3: Set the iteration counter $k = 0$
4: repeat
5:     According to function $f(\cdot)$, compute the fitness of all
       chromosomes in $P_k$
6:     Update the currently best chromosome
7:     $P_k^{'} \leftarrow$ select $N$ chromosomes using the selection operator
       on $P_k$
8:     $P_k^{''} \leftarrow$ produce new $N$ chromosomes by crossover and
       mutation operator on $P_k^{'}$
9:     $P_{k+1} \leftarrow$ perform repair operator on $P_k^{''}$
10:    $k \leftarrow k + 1$
11: until $k >$ Maximum Generation
```

4.1.1 Chromosome Representation

In MATC-GA, we hope that a solution can indicate the task-worker allocation result as well as the order in which each worker performs the tasks allocated to him. Furthermore, for different solutions, the set of workers is fixed, but the tasks assigned to each worker in each solution are uncertain. Therefore, we adopt the array structure to represent a solution chromosome. Specifically, a chromosome consists of m genetic segments where m is the number of candidate workers. The index of each genetic segment represents the ID of a worker. Each genetic segment can be further divided into several genes, the values of which represent the indices of tasks allocated to the worker. For instance, in Fig. 2, C_i and C_j are two chromosomes representing different solutions. Worker w_4 on both chromosomes are assigned with tasks t_5, t_9 and t_{10}. Note that the gene sequences reflect the execution order of the tasks. As illustrated in Fig. 2, the red solid line is the task execution order for w_4 on C_i, while the blue dotted line shows the order for w_4 on C_j.

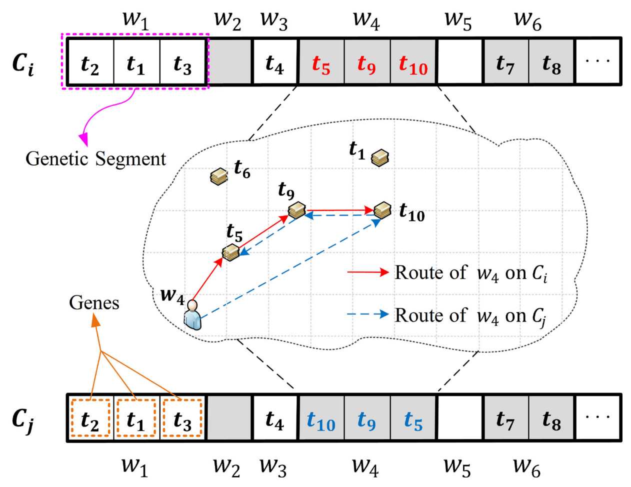

w_1 w_2 w_3 w_4 w_5 w_6
C_i
t_2 t_1 t_3 t_4 t_5 t_9 t_{10} t_7 t_8 ...
Genetic Segment
t_1 t_6 t_9 t_{10} t_5 w_4
Route of w_4 on C_i
Route of w_4 on C_j
Genes
C_j
t_2 t_1 t_3 t_4 t_{10} t_9 t_5 t_7 t_8 ...

Fig. 2. Chromosome representation.

According to Eqs. (3) and (4) in the Definition 5, we can define two types of chromosomes: valid chromosome and invalid chromosome.

- *Valid chromosome*: The allocation scheme represented by the chromosome is valid, that is, the constraints Eqs. (3) and (4) are satisfied. In addition, Eqs. (5) and (6) indicate that any task can only be found in one genetic segment, ensuring that the task can only be assigned to one worker.
- *Invalid chromosome*: The chromosome does not meet constraints Eqs. (3) and (4). Some tasks are allocated to multiple workers at the same time, violating the constraint Eqs. (5) and (6). However, invalid chromosomes may contain some good genetic segments. So it is also helpful to keep some invalid chromosomes during evolution.

Algorithm 2. Random-Greedy

```
Input: Task set $T$, Worker set $W$
Output: Chromosome $C$ (task-worker allocation scheme)
 1: Generate a chromosome $C$ with no genes
 2: Create a candidate worker set $CW \leftarrow W$
 3: Create an unallocated task set $UT \leftarrow T$
 4: repeat
 5:   Select a worker $w$ from $CW$ randomly
 6:   Set the iteration counter $k = 0$
 7:   while $k \leq$ number of tasks do
 8:     Select a task $t$ from $UT$ randomly
 9:     $C^{'} \leftarrow$ allocate $t$ to $w$ and append $t$ to the end of the task
        sequence for $w$ on $C$
10:     if $C^{'}$ is a valid chromosome then
11:       $C \leftarrow C^{'}$
12:       $UT \leftarrow UT - \{t\}$
13:     end if
14:     $k \leftarrow k + 1$
15:   end while
16:   $CW \leftarrow CW - \{w\}$
17: until $CW = \emptyset$
```

4.1.2 Population Initialization

Based on the chromosome representation, we generate the first-generation population by using Random-Greedy (Algorithm 2) to obtain the population diversity. Lines 1-3 generate an empty chromosome with no genes, a candidate worker set and a unallocated task set. Then, the main loop (Lines 4-17) iteratively selects workers and tasks randomly to get valid chromosome (task-worker allocation scheme). In Line 5, we select a worker w from the candidate worker set randomly. The inner loop (Lines 7-15) iteratively selects tasks randomly and make a decision on whether to assign tasks to the worker w based on the type of chromosome. Specifically, we select a task t that has not been allocated (Line 8), and allocate the task t to the worker w and append the task to the end of the task sequence for the worker w on the chromosome of the last iteration (Line 9). If it is a valid chromosome, the chromosome is updated and the task t is removed from the unallocated task set (Lines 10-13). In Line 16, the worker w is removed from the candidate worker set. When the candidate worker set is empty, the main loop ends.

The population of each generation is defined as: $G(i) = \{C_1, C_2, \ldots, C_k\}$, where $G(i)$ denotes the ith generation with k chromosomes and C_k denotes the kth chromosome. We only accept valid chromosomes when randomly generating the first-generation population.

4.1.3 Fitness Evaluation

In GA, the evolutionary population is evaluated on the quality of each chromosome by using the fitness function. In MATC, the objective is to maximize the utility of the MCS platform. Hence, we use the platform utility to reflect the chromosome fitness. The higher the platform utility is, the better the fitness is. Given a population $G = \{C_1, C_2, \ldots, C_k\}$, we use array $X_l = \{x_1, x_2, \ldots, x_n\}$ to represent the task allocation state for chromosome $C_l (1 \leq l \leq k)$, where the value of $x_i (1 \leq i \leq n)$ is "1" or "0". $x_i = 1$ indicates that the task t_i has been assigned, while $x_i = 0$ means that the task t_i has not been assigned. Then, the fitness of chromosome C_l can be calculated as

$$f(C_l) = \sum_{i=1}^{n} ta_i \cdot x_i. \tag{7}$$

Note that the superior chromosomes with higher fitness have higher possibilities to form a new generation. So the function $f(\cdot)$ can distinguish the quality of chromosomes.

4.1.4 Selection Operator

The selection operator is to pass the chromosomes with higher fitness to the next generation while ensuring the diversity of the population. However, some chromosomes with lower fitness may also contain some good genetic segments. In [47], the tournament selection operator has been proved to be an effective strategy for covering the population. However, the tournament selection is a stochastic operation that can lead to selection errors, and it is possible that chromosomes with higher fitness may not be available. In order to ensure that superior chromosomes can be passed on to the next generation, we adopt an enhanced selection operator with two steps to do the selection (as shown in Fig. 3).

1) Elitist preservation selection: We sort the chromosomes in the parent population according to the fitness values in the descending order, and select the first to eth (e.g., first third) of them directly to the

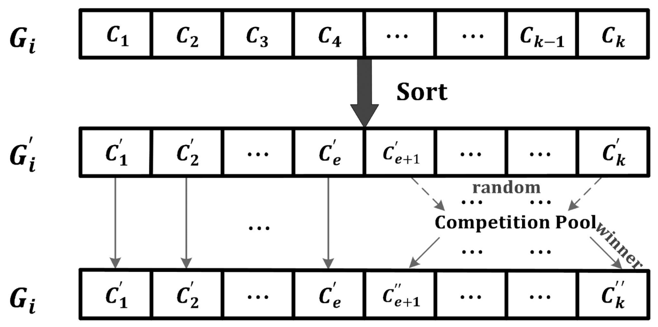

G_i
C_1 C_2 C_3 C_4 ⋯ ⋯ C_{k-1} C_k
Sort
G_i'
C_1' C_2' ⋯ C_e' C_{e+1}' ⋯ ⋯ C_k'
random
Competition Pool
winner
G_i
C_1' C_2' ⋯ C_e' C_{e+1}'' ⋯ ⋯ C_k''

Fig. 3. Selection operator.

next generation. We regard them as elitist chromosomes. This method can ensure that chromosomes with high fitness values will not be eliminated.

2) Tournament selection: For the remaining chromosomes, we regard them as ordinary chromosomes. In each operation, we randomly select a predefined number of chromosomes from ordinary chromosomes to the competition pool. Then we compare the fitness values of the chromosomes from the competition pool and pass the one with the highest fitness value to the next generation. The operation is repeated until the quantity requirement is met.

4.1.5 *Crossover Operator*

After selection, we get the most potential chromosomes. Then we need to recombine the chromosomes through crossover operation in order to produce more potential chromosomes. We use the idea of survival of the fittest to cross the corresponding genetic segments of the parent chromosomes. First of all, we need to select two chromosomes from the population as the parents for crossover operation. Specifically, we randomly select one chromosome from the set of ordinary chromosomes (such as C_i in Fig. 4), and one chromosome from the set of elite chromosomes (such as C_j in Fig. 4). This operation can make the whole population evolve and maintain the diversity of the population. Next, we generate a superior child by combining high-quality genetic segments of parents. Take the genetic segments of worker w_1 in C_i and C_j as an example (as shown in Fig. 4). In C_i, the fitness value of w_1 is $f(w_1) = 34$, while in C_j, the fitness value of w_1 is $f(w_1) = 16$. Therefore, the genetic segment of w_1 in C_i is superior and should be passed to the child chromosome C_{new}. This process is repeated until all genetic segments are compared. Note that the child chromosome produced by the crossover operator may violate the restriction conditions and leads to an invalid chromosome.

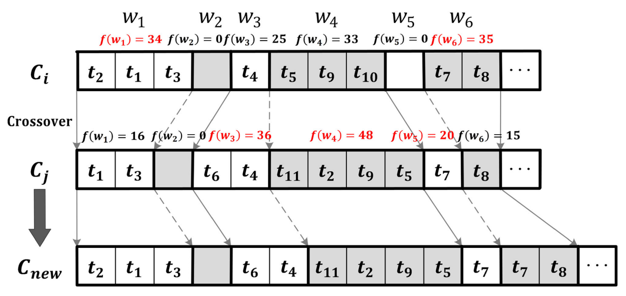

w_1 w_2 w_3 w_4 w_5 w_6
$f(w_1)=34$ $f(w_2)=0$ $f(w_3)=25$ $f(w_4)=33$ $f(w_5)=0$ $f(w_6)=35$
C_i
t_2 t_1 t_3 t_4 t_5 t_9 t_{10} t_7 t_8 ⋯
Crossover
$f(w_1)=16$ $f(w_2)=0$ $f(w_3)=36$ $f(w_4)=48$ $f(w_5)=20$ $f(w_6)=15$
C_j
t_1 t_3 t_6 t_4 t_{11} t_2 t_9 t_5 t_7 t_8 ⋯
C_{new}
t_2 t_1 t_3 t_6 t_4 t_{11} t_2 t_9 t_5 t_7 t_7 t_8 ⋯

Fig. 4. Crossover operator.

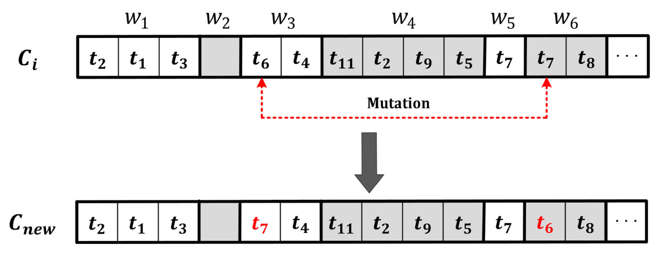

w_1 w_2 w_3 w_4 w_5 w_6
C_i
t_2 t_1 t_3 t_6 t_4 t_{11} t_2 t_9 t_5 t_7 t_7 t_8 ⋯
Mutation
C_{new}
t_2 t_1 t_3 t_7 t_4 t_{11} t_2 t_9 t_5 t_7 t_6 t_8 ⋯

Fig. 5. Mutation operator.

4.1.6 *Mutation Operator*

Mutation operator is used to change the genes of a chromosome to form a new chromosome. This operator can not only increase the diversity of the population, but also make it easier to jump out of the local optimum. For a chromosome to be mutated, we randomly select two genes in two different genetic segments. Then, the swap operation is performed on the two selected genes to generate a mutated chromosome. Note that this generated chromosome may be an invalid chromosome. An example of mutation operation is shown in Fig. 5.

Algorithm 3. Repair Operator

```
Input: An invalid chromosome $C$
Output: A valid chromosome $C_{new}$
 1: for each genetic segment $C(w)$ do
 2:   if the worker or any allocated task violates Eqs. (3) and (4) then
 3:     $C(w) \leftarrow$ the largest subset of tasks that satisfy the conditions in the genetic segment $C(w)$
 4:   end if
 5: end for
 6: for each task $t$ do
 7:   $CGS \leftarrow$ all genetic segments containing task $t$
 8:   $Cbest \leftarrow$ the genetic segment with the maximum fitness in $CGS$
 9:   for each genetic segment in $CGS - \{Cbest\}$ do
10:     Remove task $t$
11:   end for
12: end for
13: for each genetic segment $C(w)$ do
14:   Update worker $w$'s location and remaining working time by the task sequence in $C(w)$
15:   $Tw \leftarrow$ the tasks that worker $w$ can perform in unallocated tasks
16:   while $Tw \neq \emptyset$ do
17:     Select a task $t'$ from $Tw$ randomly
18:     $C'(w) \leftarrow$ add task $t'$ to the end of the task sequence $C(w)$
19:     if $C'(w)$ is a valid genetic segment then
20:       $C(w) \leftarrow C'(w)$
21:     end if
22:     $Tw \leftarrow Tw - \{t'\}$
23:   end while
24: end for
```

4.1.7 *Repair Operator*

In order to make an invalid chromosome generated during crossover and mutation become a valid chromosome, we design a repair operator (Algorithm 3). According to the

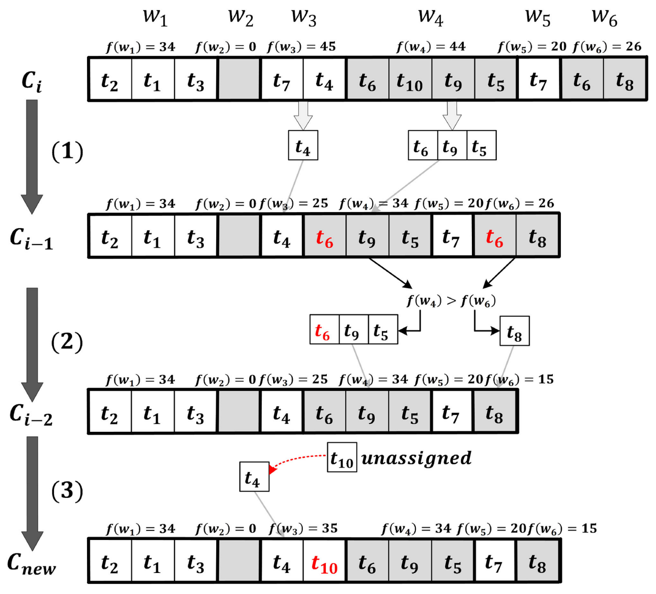

w_1 w_2 w_3 w_4 w_5 w_6
$f(w_1)=34$ $f(w_2)=0$ $f(w_3)=45$ $f(w_4)=44$ $f(w_5)=20$ $f(w_6)=26$
C_i
(1)
C_{i-1}
$f(w_1)=34$ $f(w_2)=0$ $f(w_3)=25$ $f(w_4)=34$ $f(w_5)=20$ $f(w_6)=26$
$f(w_4)>f(w_6)$
(2)
C_{i-2}
$f(w_1)=34$ $f(w_2)=0$ $f(w_3)=25$ $f(w_4)=34$ $f(w_5)=20$ $f(w_6)=15$
unassigned
(3)
C_{new}
$f(w_1)=34$ $f(w_2)=0$ $f(w_3)=35$ $f(w_4)=34$ $f(w_5)=20$ $f(w_6)=15$

Fig. 6. Repair operator.

constraints given by Eqs. (3) and (4), both workers and tasks need to satisfy their own time constraints. In addition, the same task cannot be allocated to more than one worker simultaneously according to Eqs. (5) and (6). Similar to the previous defined types of chromosomes, we define the types of genetic segments: 1) valid genetic segment: the worker and all allocated tasks satisfy the time constraints given by Eqs. (3) and (4); 2) invalid genetic segment: the worker or any allocated task violates the time constraints given by Eqs. (3) and (4). Then, we repair an invalid chromosome by the following steps:

1) Check each genetic segment (Lines 1-5) to verify (a) whether the worker can complete the tasks according to the prescribed order within the expected working time, and (b) whether the allocated tasks can be completed by the worker before the task expiration time. If any one of the above two conditions is violated, we search for the largest subset of tasks that satisfy the conditions in the current genetic segment as a new allocation scheme. Take worker w_3 in Fig. 6-(1) as an example. We assume that the current allocation cannot meet one of the above two conditions. Then we need to find a subset satisfying both conditions and having the largest fitness value from $\{\{\varnothing\},\{t_7\},\{t_4\},\{t_7,t_4\}\}$.
2) Find the task which is allocated more than once in the chromosome (Line 7). Then, the fitness values of the genetic segments that contain the same task are compared (Lines 8). We retain the task in the genetic segment with the maximum fitness value and remove it from other genetic segments (Lines 9-11). An example is shown in Fig. 6-(2).
3) After the first two steps, we obtain a valid chromosome. However, some workers may have enough time to complete some unassigned tasks. So we redistribute the remaining unallocated tasks as many as possible given that the new chromosome satisfies the constraints given by Eqs. (3), (4), (5) and (6) (Lines 13-24). An example is shown in Fig. 6-(3).

4.1.8 *Complexity Analysis of MATC-GA*

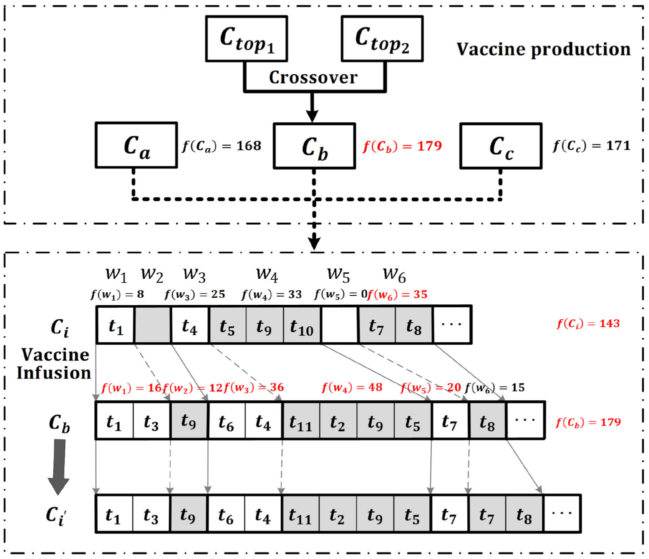
The parameters that can be considered as constants in MATC-GA are the number of chromosomes in the population k and the maximum generation MG. Given the worker set W with m workers and the task set T with n tasks, the computation complexity of the population initialization, selection operator, crossover operator and mutation operator is $\mathcal{O}(k\times m\times n)+\mathcal{O}(2k)+\mathcal{O}(k\times m)+\mathcal{O}(k)=\mathcal{O}(m\times n)$. Next, we analyze the computation complexity of the three steps in the repair operator. First, the computation complexity of finding all ordered subsets of an ordered set is $\mathcal{O}(m\times T_w)$, where T_w is the set of tasks assigned to worker w, and its size will not exceed n. Second, the computation complexity of repairing the task which is allocated more than once in the chromosome is about $\mathcal{O}(n)$. Finally, the number of tasks that the worker with updated information can perform in unallocated tasks is less than n, so the computation complexity will not exceed $\mathcal{O}(m\times n)$. Therefore, the overall computation complexity of MATC-GA is about $MG\times(\mathcal{O}(m\times n)+\mathcal{O}(m\times T_w)+\mathcal{O}(n)+\mathcal{O}(m\times n))=\mathcal{O}(m\times n)$.

Algorithm 4. MATC-IGA

```
Input: Task set $T$, Worker set $W$, Number of intermediate individuals $M$, proportion of vaccine immunity $\beta$
Output: Best chromosome (task-worker allocation scheme)
1: Initialize population $G(0)$ with $N$ chromosomes
2: $P_0 \leftarrow G(0)$
3: Set the iteration counter $k=0$
4: repeat
5:    Compute the fitness of all chromosomes in $P_k$ according to the function $f(\cdot)$
6:    Update the currently best chromosome
7:    $V \leftarrow$ make the vaccines
8:    $S_k \leftarrow$ select $M$ chromosomes using the IGA_Selection operator
9:    $P_k^{'} \leftarrow$ select $\beta\cdot M$ chromosomes from $S_k$ for vaccine infusion operator
10:   $P_k^{''} \leftarrow$ produce new $M$ chromosomes by crossover on $S_k$
11:   $P_k^{'''} \leftarrow$ produce new $M+\beta\cdot M$ chromosomes by mutation on $P_k^{'}\cup P_k^{''}$
12:   $R_k \leftarrow$ perform repair operator on $P_k^{'''}$
13:   $P_{k+1} \leftarrow$ sort $R_k$ in descending order with respect to the fitness values, and select the first $N$ chromosomes
14:   $k \leftarrow k+1$
15: until $k > Maximum\ Generation$
```

4.2 Immune Genetic Algorithm for MATC

In addition to the population evolution nature of MATC-GA, we investigate to incorporate the individual learning feature into the GA-based algorithms. Specifically, we propose the *immune genetic algorithm for MATC (MATC-IGA)* by introducing the immune principle and the vaccine method, which is able to weaken the unfavorable influence caused by random selection and mutation operation. In addition, MATC-IGA can solve the problems of early maturity of the population, early convergence into a local optimal solution, and difficulty in obtaining a stable solution. The pseudocode of MATC-IGA is shown in Algorithm 4. Different from MATC-GA, we have added the steps for vaccine production (Line 7) and infusion (Line 9). During the evolution, we



C_{top_1}
C_{top_2}
Vaccine production
Crossover
C_a
$f(C_a) = 168$
C_b
$f(C_b) = 179$
C_c
$f(C_c) = 171$
w_1 w_2 w_3 w_4 w_5 w_6
$f(w_1) = 8$ $f(w_3) = 25$ $f(w_4) = 33$ $f(w_5) = 0$ $f(w_6) = 35$
C_i
t_1 t_4 t_5 t_9 t_{10} t_7 t_8 ⋯
$f(C_i) = 143$
Vaccine Infusion
$f(w_1) = 16$ $f(w_2) = 12$ $f(w_3) = 36$ $f(w_4) = 48$ $f(w_5) = 20$ $f(w_6) = 15$
C_b
t_1 t_3 t_9 t_6 t_4 t_{11} t_2 t_9 t_5 t_7 t_8 ⋯
$f(C_b) = 179$
C_i'
t_1 t_3 t_9 t_6 t_4 t_{11} t_2 t_9 t_5 t_7 t_7 t_8 ⋯

Fig. 7. Vaccine production and vaccine infusion.

increase the number of chromosomes in the population so that the diversity and stability of the population can be improved. In MATC-IGA, chromosome representation, population initialization, fitness evaluation and mutation operator are the same as MATC-GA; while crossover operator randomly selects two chromosomes that have not been crossed yet in the population, rather than selecting between elite chromosomes and ordinary chromosomes as MATC-GA. The details of new operations are as follows:

4.2.1 Vaccine Production

Unlike the traditional elitism used by GA, the vaccination strategy learns the excellent genetic segments of all chromosomes in the population and combines them to obtain a vaccine (an invalid chromosome), while the traditional elitism selects an optimal chromosome (a valid chromosome) in the population. Specifically, the vaccine is a special chromosome which retains characteristics of superior chromosomes in a generation. We infuse the vaccine into other chromosomes, so that we can get better chromosomes and the population does not degenerate under the premise of maintaining diversity. Specifically, as shown in Fig. 7, in the kth generation, we select the top two chromosomes C_{top_1} and C_{top_2} with the highest fitness values to perform crossover and repair operations to obtain the candidate vaccine C_b. Then, we compare the fitness values of the chromosome with the highest fitness value C_a, the candidate vaccine C_b, and the vaccine C_c in the $(k-1)$th generation. We choose the one with the highest fitness value as the new vaccine for the current generation.

4.2.2 IGA_Selection Operator

In the kth generation, we select all the N chromosomes directly as part of the intermediate $M (M \geq N)$ chromosomes. The remaining $M - N$ chromosomes are randomly selected from the N chromosomes by the roulette wheel [48]. The reason we do not use the roulette wheel to select the M intermediate chromosomes directly is that if we select them randomly, some chromosomes may be selected multiple times and some others may not be selected, which results in the reduction of the population diversity. Furthermore, selecting a chromosome too often may lead to invalid crossover in the subsequent crossover operation.

4.2.3 Vaccine Infusion Operator

The vaccine infusion shown in Fig. 7 is to crossover the vaccine C_b with the selected chromosomes such as C_i to ensure that the excellent genes in the vaccine can be passed on to the next generation. The difference between the vaccine infusion operator and the crossover operator is that the parent chromosomes are the chromosomes selected for vaccine infusion and the vaccine.

4.2.4 Complexity Analysis of MATC-IGA

Different from MATC-GA, the steps of vaccine production and infusion are added in MATC-IGA. The computation complexity of these two steps is $\mathcal{O}(m) + \mathcal{O}(m) = \mathcal{O}(m)$. The other steps of MATC-IGA are the same as MATC-GA. Therefore, the overall computation complexity of MATC-IGA is $\mathcal{O}(m \times n)$.

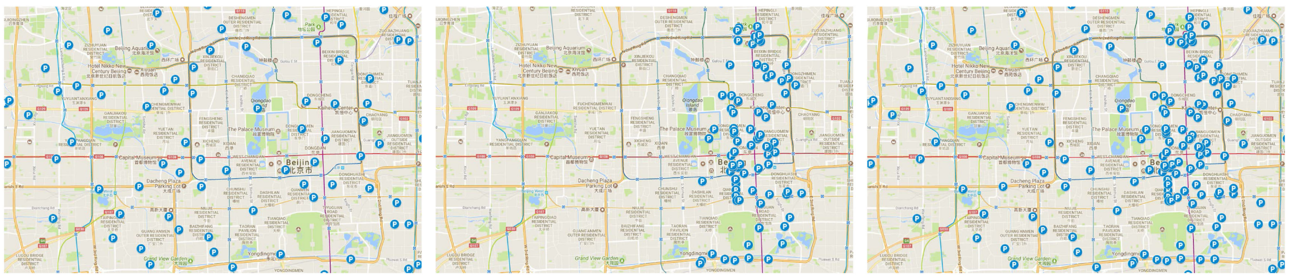
5 Experiment

In this section, we evaluate the performance of the proposed algorithms based on the synthetic and real datasets. We first introduce the datasets and the compared algorithms, and then discuss the experimental results in detail. In addition, we develop a prototype to measure the performance of the system in a real-world environment. We will discuss the result of running the prototype briefly.

5.1 Datasets

5.1.1 Synthetic Datasets

We simulate three types of task distributions to measure the scalability and flexibility of the model and algorithms (Fig. 8):



(a) Uniform distribution (b) Compact distribution (c) Mixed distribution

Fig. 8. The three types of task distributions.

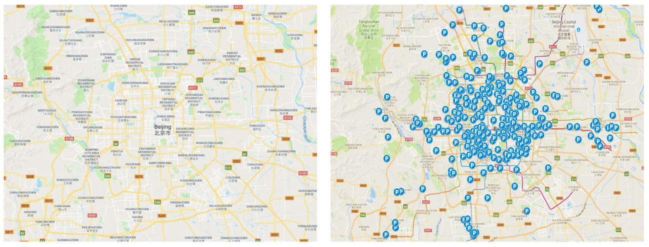

(a) Map of Beijing (b) Some parking places of Beijing

Fig. 9. Parking places of Beijing.

- *Uniform Distribution*: The uniform distribution can be used when the tasks that are dispersed in the sensing area. For instance, generating a noise map may require location-based query tasks with uniform distribution over the sensing area.
- *Compact Distribution*: In the compact distribution, the tasks are mostly concentrated in a small area of the sensing area, and the distance between each task is relatively small. This type of tasks can occur in areas where there are large-scale local events and crowds, like the Olympic Games.
- *Mixed Distribution*: The mixed distribution is a mixture of the uniform and compact distributions. Many tasks conform to the mixed distribution, such as reporting traffic conditions on different roads during rush hours. The roads in the bustling areas require more sensed data than the roads in other areas (such as residential and rural areas).

We generate workers in the sensing area randomly. The position coordinates of the tasks and workers are restricted to [0,50]. We assume that each worker has a unit velocity, such that the traveling time of a worker directly reflects his traveling distance. Then, the maximum traveling time of each worker is randomly generated within the range of [5,15], and the valid time of each task is randomly set within the range of [2,15]. Finally, we set the utility of each task within the range of [5,30].

5.1.2 Real-World Datasets

Next, we test the algorithms by using the real-world datasets, T-drive [49], [50] and parking places of Beijing [51]. T-drive consists of taxi trajectories which was generated by 10,357 taxis in a period of one week in Beijing. Each trajectory record represents the travel history of a taxi, including the taxi ID, date, time, and location (latitude and longitude). We regard each taxi as a worker and take his historic initial location along a trajectory as the starting location for undertaking sensing tasks. We set the moving velocity of a worker to 35 km/h according to the computation in [52]. In addition, we randomly select a period of continuous driving time of a worker in the trajectory as the expected working time. Continuous driving can be judged by whether the distance between two consecutive GPS sampling points is greater than 100 meters.

Another real-world dataset, parking places of Beijing, contains 5,881 parking places of Beijing. The map of Beijing and some parking places of Beijing are shown in Fig. 9. Each parking place is associated with an id, address, latitude, and longitude, and we consider the parking places as the locations of sensing tasks. The valid time of each task is randomly generated within the range of [2,15] minutes, and the value of each task is set to be within the range of [5,30].

Note that the aforementioned necessary parameters in both synthetic and real-world datasets are generated by ensuring that the following unreasonable situations will not occur:

- A worker works too long, so that he can complete most of the tasks;
- Expected working time of most workers is too short, so that few tasks can be completed;
- The valid time of most tasks is too short, so that few workers can complete the tasks;
- The valid time of most tasks is so long that it violates the temporal features of many location-based tasks.

5.2 Compared Algorithms

Greedy. The first compared algorithm uses the greedy heuristic. Specifically, The algorithm chooses the closest task to determine whether it can be assigned to a candidate worker. If the assignment is valid, the location of the assigned task is considered as the new starting location of the worker. The assignment process continuous until the worker's expected working time is exhausted.

GGA-I. The second compared algorithm is the greedy-enhanced genetic algorithm for intentional movement (GGA-I) proposed in [19]. The initial population of GGA-I is generated using the result of the NearestFirst algorithm which is a kind of greedy heuristic. Due to the difference of the solution representation, we adapt the operation constraints to meet the proposed optimization model.

DAEA. The third compared algorithm is an exact algorithm for destination-aware spatial crowdsourcing (named DAEA here) from [21]. Since DAEA is an exact algorithm, we only compare it in small-scale synthetic datasets.

EA. In small-scale datasets, we also find optimal solutions for comparison by running an exact algorithm (named EA) based on the branch and bound method [53], [54].

5.3 Parameter Settings for GA

The main parameters concerning GA include the population size N, the crossover probability P_c, the mutation probability P_m, proportion of vaccine immunity β, and the maximum generation MG. We set $N = 50, P_c = 0.9, P_m = 0.01, \beta = 0.1$ following the common settings for these parameters. Then through the experiment by setting $m = 60, n = 200$ in the synthetic dataset (as shown in Fig. 10), we can observe that MATC-IGA and MATC-GA start to converge when the number of iterations (i.e., the maximum generation MG) is around 22. Considering that the common setting for the maximum generation is around 100, we set $MG = 100$ in our experiments.

5.4 Results

In the experiment, we evaluate the performance of the algorithms with respect to the utility of the MCS platform, the number of allocated tasks (the ratio of allocated tasks), and the average number of tasks allocated to each worker. The utility of the MCS platform is a direct reflection of the

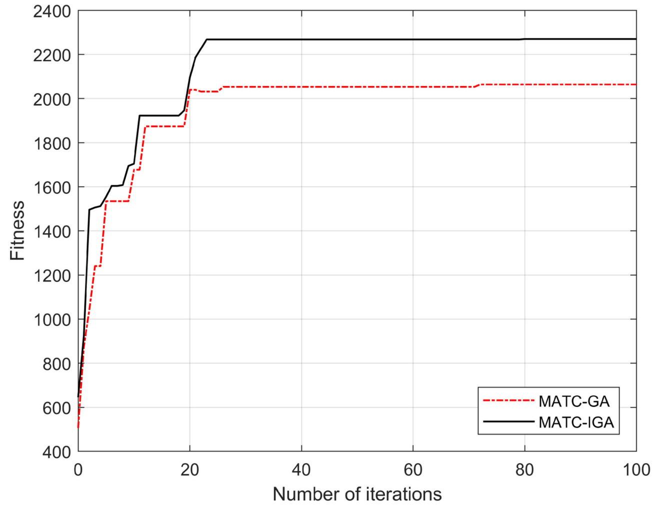


Fig. 10. Convergence of MATC-GA and MATC-IGA.

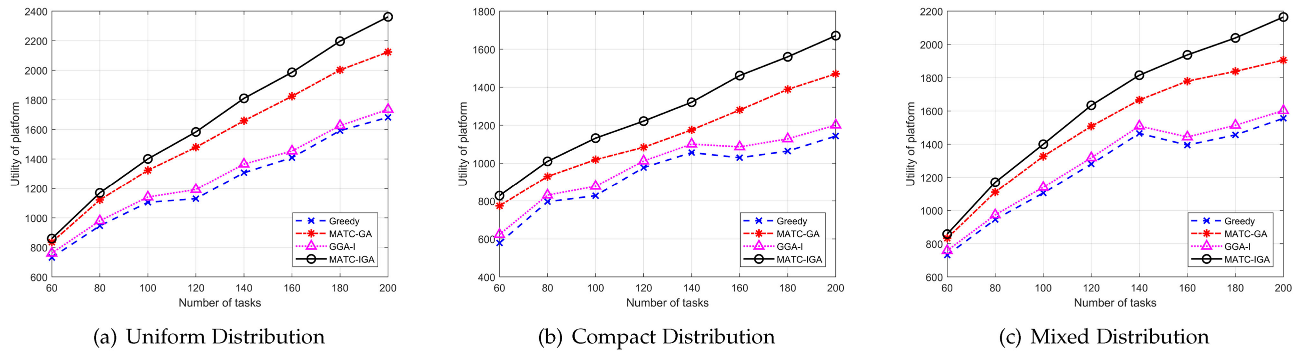
optimization goal, and higher utility indicates better results. For task requesters, the high allocation rate of the tasks will attract more task requesters; while for workers, the more tasks a worker completes, the more reward he can receive. For each experiment, the algorithms run 50 times to get the average values. All the algorithms are implemented on an Intel Core i7-7700 CPU @3.60 GHz with 8 GB RAM.

5.4.1 The Effect of the Task Number in Synthetic Datasets

We fix the number of workers $m = 60$ and set the number of tasks $n = \{60, 80, 100, 120, 140, 160, 180, 200\}$. From Figs. 11 and 12, we can see that with the increment of the number of tasks, the utility of the MCS platform increases, while the ratio of allocated tasks decreases. This trend is due to the fact that, as the number of tasks increases, the workers have a higher probability to be assigned with the tasks that are more beneficial to the platform. At the same time, due to the limited number of workers and their limited working time, only part of the tasks can be accomplished. As a result, with the increment of the number of tasks, more and more tasks will not be allocated, which leads to a reduction in the ratio of allocated tasks.

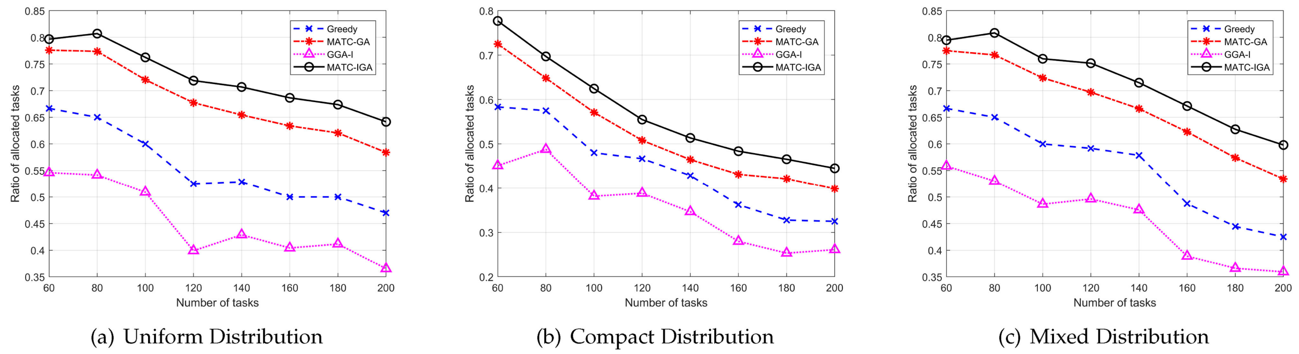
Fig. 11 shows the comparison of the platform utility under different task distributions. It can be observed that the different algorithms all perform better under the uniform distribution, and perform worse under the compact distribution. This is because that, tasks with the uniform distribution are accessible to more workers in the sensing area, but tasks with the compact distribution may leave the workers who are far from the tasks idle. Next, we illustrate the numerical performance comparison of different algorithms. In the three types of distributions, the proposed MATC-GA and MATC-IGA achieve the highest utility values. Specifically, as shown in Table 3, the utility value of MATC-GA is around 17.28 and 22.10 percent larger on average than that of GGA-I and Greedy, respectively; while the utility value of MATC-IGA is around 8.97 percent larger on average than that of MATC-GA. In addition, since one worker can accomplish multiple tasks, the performances of the four algorithms are similar when the number of tasks and workers are comparable.

On the other hand, the ratios of allocated tasks of the four algorithms decrease with the increment of task numbers (as shown in Fig. 12). It can be observed that the ratio of allocated tasks for the compact distribution is less satisfactory.



(a) Uniform Distribution (b) Compact Distribution (c) Mixed Distribution

Fig. 11. Utility of the platform ($m = 60$).



(a) Uniform Distribution (b) Compact Distribution (c) Mixed Distribution

Fig. 12. Ratio of the allocated tasks ($m = 60$).

TABLE 3
Average Performance Comparison of the Platform Utility (w.r.t. the Number of Tasks)

Algorithms	*Greedy*	*GGA-I*	*MATC-GA*	*MATC-IGA*
Greedy	–	4.09%	22.10%	33.10%
GGA-I	-3.91%	–	17.28%	27.85%
MATC-GA	-17.86%	-14.53%	–	8.97%
MATC-IGA	-24.52%	-21.47%	-8.15%	–

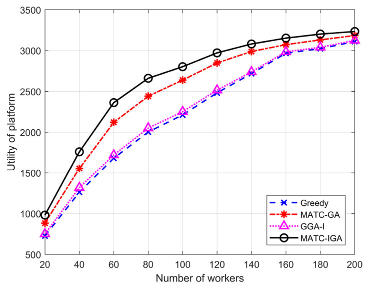
TABLE 4
Average Performance Comparison of the Ratio of the Allocated Tasks (w.r.t. the Number of Tasks)

Algorithms	*Greedy*	*GGA-I*	*MATC-GA*	*MATC-IGA*
Greedy	–	-18.85%	20.72%	30.24%
GGA-I	23.35%	–	49.01%	60.79%
MATC-GA	-16.96%	-32.57%	–	7.86%
MATC-IGA	-22.96%	-37.42%	-7.24%	–

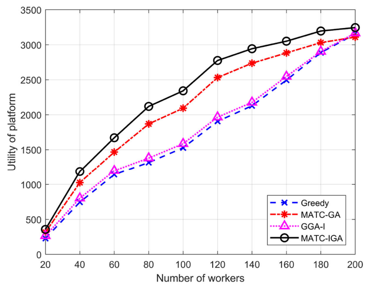
As the number of tasks increases, the ratio of allocated tasks drops significantly, and the MATC-IGA algorithm with the best performance also drops below 50 percent. As listed in Table 4, MATC-GA and MATC-IGA perform better than GGA-I and Greedy, with average improvement ratios of 49.01/20.72 and 60.79/30.24 percent, respectively.

5.4.2 *The Effect of the Worker Number in Synthetic Datasets*

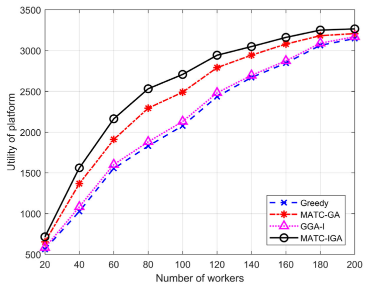
Next, we fix the number of tasks $n = 200$ and set the number of workers $m = \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$ to study the effect of the worker number. In Figs. 13, 14, and 15, there are three types of results: the utility of the MCS platform, the number of allocated tasks, and the average number of tasks allocated to each worker. Through the first two types of results, we can see that with the increment of the number of workers, both the utility of the platform and the number of allocated tasks exhibit an upward trend. The reason is that as the number of workers increases, the candidate pool of workers is larger so that the possibility for tasks to be allocated is increased. On the other hand, for the three different task distributions, when the number of tasks and workers are comparable (for example, $n = 200$ and



(a) Uniform Distribution

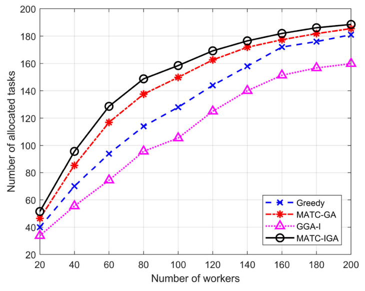


(b) Compact Distribution

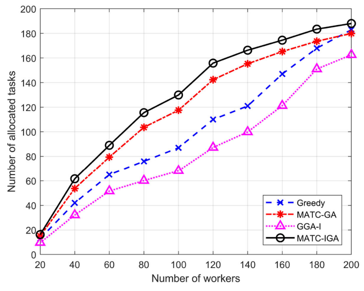


(c) Mixed Distribution

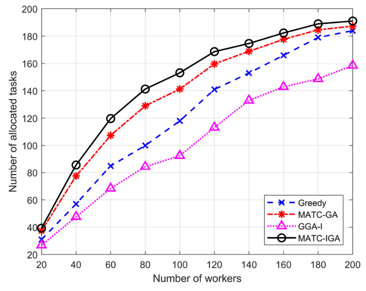
Fig. 13. Utility of the platform ($n = 200$).



(a) Uniform Distribution

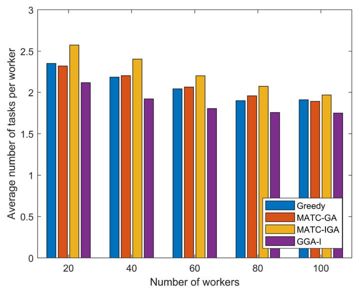


(b) Compact Distribution

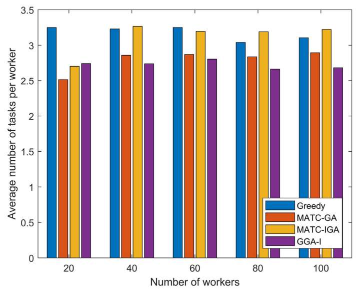


(c) Mixed Distribution

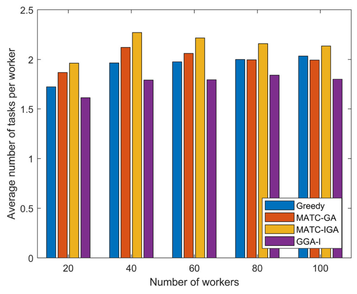
Fig. 14. Number of the allocated tasks ($n = 200$).



(a) Uniform Distribution



(b) Compact Distribution



(c) Mixed Distribution

Fig. 15. Average number of tasks per worker ($n = 200$).

TABLE 5
Average Performance Comparison of the Platform Utility (w.r.t. the Number of Workers)

Algorithms	*Greedy*	*GGA-I*	*MATC-GA*	*MATC-IGA*
Greedy	–	2.88%	18.85%	28.70%
GGA-I	-2.72%	–	15.38%	24.84%
MATC-GA	-14.97%	-12.70%	–	7.88%
MATC-IGA	-20.76%	-18.71%	-7.16%	–

$m = [140, 200]$), the results obtained by different algorithms in the three task distributions are not significantly different. However, when the number of workers and the number of tasks have larger differences (for example, $n = 200$ and $m = [20, 120]$), the performances of different algorithms under the uniform distribution are much better than that under the compact distribution, which is consistent with the aforementioned conclusion.

We then analyze the average performances of the algorithms under the three task distributions. For the utility of the platform (as shown in Fig. 13), the performance of MATC-IGA perform best with different numbers of workers, followed by MATC-GA. Specifically, as shown in Table 5, MATC-IGA is better than MATC-GA, GGA-I and Greedy with average increment ratios of around 7.88, 24.84 and 28.70 percent, respectively; while MATC-GA is better than GGA-I and Greedy with average increment ratios of around 15.38 and 18.85 percent, respectively. Furthermore, when the ratio of the number of tasks to the number of workers is about 4:1, the performance of MATC-IGA and MATC-GA can be optimized. Similarly, for the number of allocated tasks (as shown in Fig. 14), MATC-IGA is still better than the other algorithms. In contrast, GGA-I has the worst performance. Specifically, as shown in Table 6, the average increment ratio of MATC-IGA is up to 6.99/51.80/25.44 percent compared to that of MATC-GA/GGA-I/Greedy, respectively.

Finally, we analyze the third type of results in detail, where the average number of assigned tasks per worker is used to measure the performance of the algorithms. If the number of assigned tasks per worker is higher, it means that workers can get higher reward. While maximizing the utility of the platform, we can achieve a win-win situation between the MCS platform and the workers if we can assign more tasks for each worker. When the number of workers m gradually increases (gradually becoming commensurate with the number of tasks), there is a greater possibility a fixed number of tasks can be allocated. In this case, all of the algorithms can achieve better performance. Here we set $m = \{20, 40, 60, 80, 100\}$ to investigate the situation when the number of workers is relatively small. First we analyze the tasks with the uniform distribution (as shown in Fig. 15a). It can be observed from the figure that as the number of workers increases, the average number of tasks assigned per worker decreases slightly. MATC-IGA algorithm has the best results. The results of MATC-GA and Greedy are similar and are slightly worse than that of MATC-IGA, while the results of GGA-I is the worst. Considering the compact distribution as shown in Fig. 15b, it is not difficult to find that the results are different from those in the uniform distribution. The reason is that because of the compact distribution of tasks, a worker can spend less time in traveling and undertake more tasks, which happens to reflect the local search nature of Greedy. With the increment of the number of workers, MATC-IGA can achieve similar results compared with Greedy. For the mixed distribution, as shown in Fig. 15c, it is clear that MATC-IGA is better than the other algorithms, following by MATC-GA and Greedy. In conclusion, MATC-IGA outperforms the other algorithms in all three distributions, while MATC-GA and Greedy have similar performances. Combined with the first two metrics, MATC-IGA has the best overall performance, followed by MATC-GA. It confirms the effectiveness and feasibility of the proposed algorithm.

TABLE 6
Average Performance Comparison of the Ratio of the Allocated Tasks (w.r.t. the Number of Workers)

Algorithms	*Greedy*	*GGA-I*	*MATC-GA*	*MATC-IGA*
Greedy	–	-16.79%	16.93%	25.44%
GGA-I	20.49%	–	41.33%	51.80%
MATC-GA	-13.71%	-27.98%	–	6.99%
MATC-IGA	-19.03%	-32.33%	-6.42%	–

TABLE 7
Average Time Complexity of MATC-GA and MATC-IGA [w.r.t. Runtime per Generation(s)]

m = 60	Number of tasks				
	60	80	100	120	140
MATC-GA	0.1479	0.2059	0.2978	0.3938	0.4936
MATC-IGA	0.2379	0.3154	0.4468	0.5927	0.7458
n = 200	**Number of workers**				
	60	80	100	120	140
MATC-GA	0.7268	0.7816	0.8224	0.8746	0.9189
MATC-IGA	1.0914	1.1045	1.2246	1.3167	1.3904

5.4.3 The Time Complexity of MATC-GA and MATC-IGA in Synthetic Datasets

We have already analyzed the computation complexity of MATC-GA and MATC-IGA in the previous section. Although the solution space of MATC grows exponentially with the increasing number of tasks and workers, the computation complexity MATC-GA and MATC-IGA does not increase dramatically, as shown in Table 7. When the number of tasks (workers) is constant, the time complexity increases linearly as the increment of workers (tasks). Note that because the time complexity of MATC-GA and MATC-IGA is affected by the number of iterations, we compare the CPU time of each iteration.

5.4.4 Results in Small-Scale Synthetic Datasets

In order to compare the performance of the proposed algorithms with exact algorithms, we reduce the number of tasks and workers for small-scale experiments. First, we fix the number of workers $m = 35$ and set the number of tasks $n = \{35, 40, 45, 50, 55, 60, 65, 70, 75, 80\}$ to get the performance of the exact algorithms and the proposed algorithms in three task distributions. Due to the limited space, we only show the result figures of the compact distribution (Fig. 16). The figures of the other distributions show similar trends

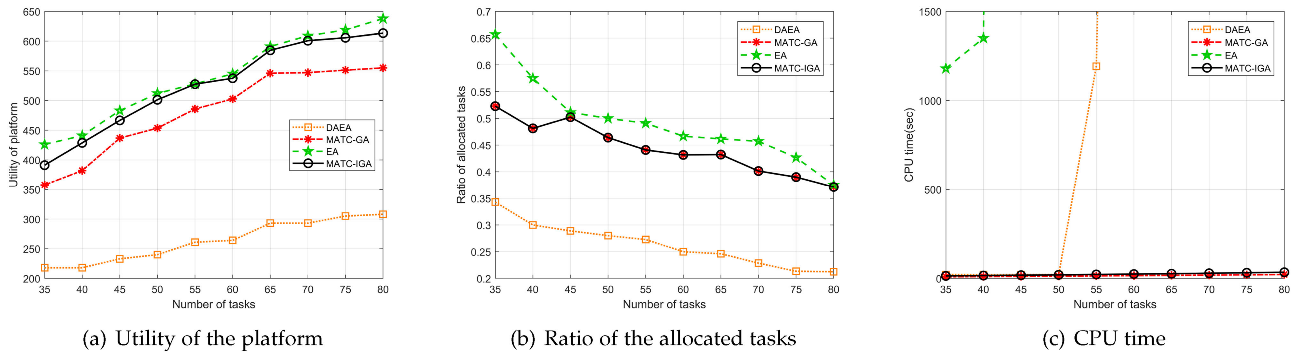


Fig. 16. The effect of the number of tasks in small-scale dataset with compact distribution ($m = 35$).

and are presented in Appendix B, available in the online supplemental material. From Figs. 16a and 16b, we can see that the trends of the algorithms are similar to that in the aforementioned results. Furthermore, we can observe from Fig. 16c that when the number of workers is fixed, the CPU time of the proposed algorithms grows linearly with the increment of the number of tasks, which is consistent with the conclusions obtained from the previous analysis and experiments. On the contrary, the exact algorithms (EA and DAEA) are limited by the scale of problem. As the number of tasks increases to certain extent, the time grows sharply, so that they cannot be applied to large-scale problems.

Next, we illustrate the numerical performance comparison of different algorithms. In the three types of distributions, the proposed MATC-GA and MATC-IGA can achieve a superior solution close to the optimal solution. Specifically, the average utility values of MATC-IGA and MATC-GA can achieve 97.32 and 91.75 percent of the value of EA respectively, while DAEA can only achieve 59.7 percent. On the other hand, MATC-IGA, MATC-GA and DAEA can achieve 93.91, 93.91 and 63.14 percent in the ratio of allocated tasks obtained by EA, respectively. Here we can find that MATC-IGA can achieve higher utility of the platform when the ratios of allocated tasks of MATC-IGA and MATC-GA are the same. It indicates that the immune principle and the vaccine method are able to weaken the unfavorable influence caused by the random selection and the mutation operation.

Considering the effect of the worker number in small-scale synthetic datasets, we fix the number of tasks $n = 50$ and set the number of workers $m = \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$. Similarly, we only show the result figures of the compact distribution (Fig. 17). The figures of the other distributions show similar trends and are presented in Appendix C, available in the online supplemental material. The trends shown in Figs. 17a and 17b are similar to that in the aforementioned results with a larger synthetic dataset. Fig. 17c shows that, with the increment of workers, the running time of the proposed algorithms increase linearly, while EA and DAEA grows sharply when the number of workers increases to certain extent.

We now summarize the average performance of the algorithms under the three task distributions. For the utility of the platform, MATC-IGA and MATC-GA perform well with different numbers of workers. Specifically, MATC-IGA and MATC-GA can reach 97.37 and 92.18 percent of the optimal value returned by EA on average, while DAEA can reach 52.42 percent of the optimal by EA. Similarly, for the number of allocated tasks, both MATC-IGA and MATC-GA can reach 92.85 percent of the result by EA. In contrast, DAEA has the worst performance, which can only reach 56.71 percent of the result by EA.

See Appendix B and C, available in the online supplemental material, for details of the results in small-scale synthetic datasets.

5.4.5 *The Results in Real-World Datasets*

We also compare the algorithms based on the real-world datasets. In order to test the applicability of the algorithms, we randomly select 200 workers and 200 tasks based on the real-world datasets to perform the experiments.

First, we analyze the effect of task numbers. We set the number of workers $m = 60$ and the number of tasks $n = \{60, 80, 100, 120, 140, 160, 180, 200\}$. As shown in Fig. 18a, we can see that the trend presented by the results is similar to that produced by the previous synthetic data. Specifically, the

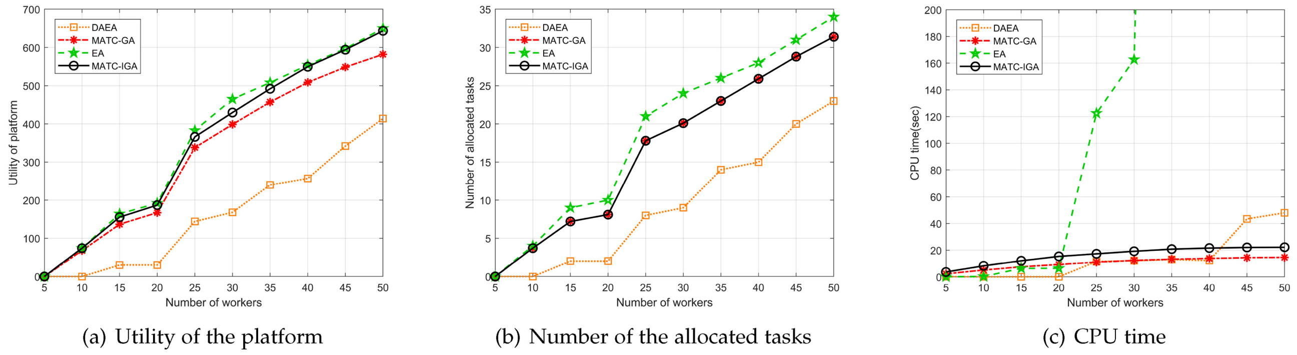
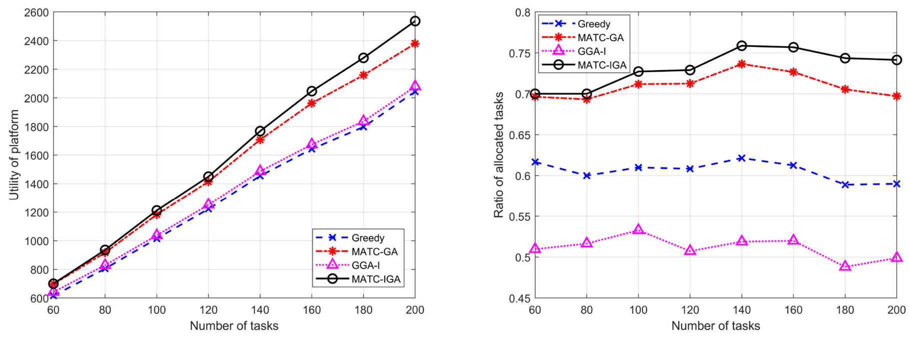


Fig. 17. The effect of the number of workers in small-scale dataset with compact distribution ($n = 50$).



(a) Utility of the platform (b) Ratio of allocated tasks

Fig. 18. The effect of the task number in real-world datasets.

utility of platform obtained by MATC-IGA is better than GGA-I and Greedy (9.00 to 24.13 percent better than GGA-I, 13.63 to 26.84 percent better than Greedy), and is similar to MATC-GA. But for the ratio of allocated tasks (as shown in Fig. 18b), we can see that the result is not the same as what we get from experiments with synthetic data. The results show that for each algorithm, the ratio of allocated tasks does not change significantly with increment the number of tasks. Instead, the ratios of allocated tasks keep almost at the same values (73 percent for MATC-IGA, 71 percent for MATC-GA, 50 percent for GGA-I, 60 percent for Greedy). In other words, even if the resources of workers are limited, some newly added tasks can still be assigned to the appropriate workers. This phenomenon is because that, in the real data, the workers are not uniformly distributed, which leads to the allocation becoming more random and the result fluctuates in a small range rather than in a downward trend.

Next, we analyze the effect of worker numbers. As shown in Figs. 19a and 19b, we can see that the results are similar to the overall trend of the results obtained from the previous synthetic data. The only difference is that the trend of the results obtained here is steep first and becomes flat gradually, while in the synthetic data, the growth trend is steadily rising. These results are also caused by the nonuniform distribution of workers. Numerically, the utility of the platform obtained by the MATC-IGA is at least 7.98 percent higher than that obtained by GGA-I, and the highest increment reaches 50.86 percent. Compared with Greedy, the utility of MATC-IGA is 8.91 to 59.56 percent higher. The average gap between MATC-GA and MATC-IGA is about 4.37 percent. On the other hand, MATC-GA and MATC-IGA achieve considerable numbers of allocated tasks. Specifically, compared with GGA-I and Greedy, the improvement of MATC-GA is more than 22.81 and 7.21 percent at least, and more than 37.85 and 16.06 percent on average. MATC-IGA is shown to be slightly better than MATC-GA, with an improvement of around 0.91 to 9.67 percent. As for the average number of tasks allocated per worker (as shown in Fig. 19c), both MATC-IGA and MATC-GA have high competitiveness similar to the above conclusions.

To sum up, through these three measures, the proposed algorithms MATC-IGA and MATC-GA achieve more competitive and stable performances compared with GGA-I and Greedy.

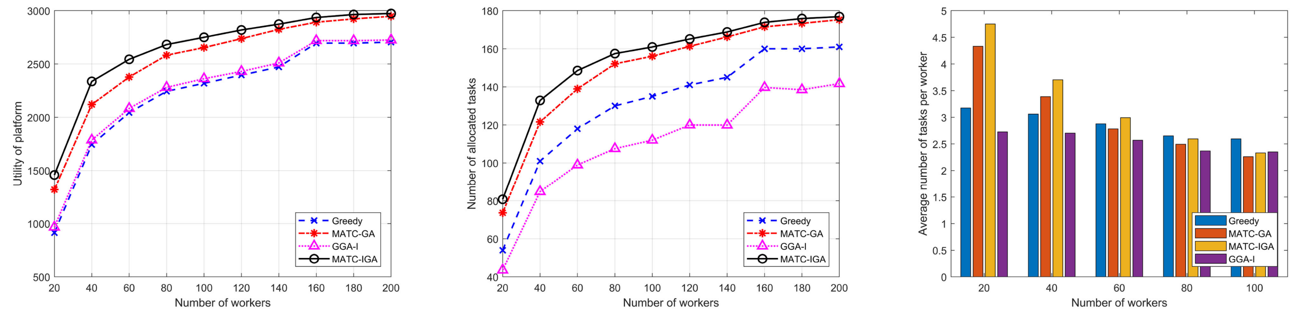
5.5 A Crowdsensing Prototype for MATC

In order to measure the real performance of the proposed algorithms and MCS system, we deploy a prototype named MatcSense in a limited experimental environment. MatcSense is used to monitor garbage classification in various areas of the campus by uploading photos of garbage areas in campus. The preliminary result of running MatcSense show that, our proposed algorithms can effectively find the superior solution, which also proves the effectiveness and applicability of our algorithms. Due to the page limit of the paper, the details about the prototype is placed in Appendix D, available in the online supplemental material.

6 Discussion

This section discusses the limitations of the studied model and the proposed algorithms.

1) *Efficiency*: The proposed algorithms may be less efficient in a very large scale environment (e.g., a city-wide



(a) Utility of the platform (b) Number of allocated tasks (c) Average number of tasks per person

Fig. 19. The effect of the worker number in real-world datasets.

application). In this case, the combination of distributed technology such as fog computing and mobile crowdsensing is promising for effectively solving this problem and thus can further improve the allocation efficiency. Recently, researchers have begun to pay attention to incorporate the concept of fog computing to task allocation in mobile crowdsensing [55], which focuses on secure data deduplication scheme to improve communication efficiency while guaranteeing data confidentiality. How to coordinate distributed fog nodes to design more efficient allocation strategies is still a challenging and unsolved problem in MCS.

2) *Scalability*: The proposed algorithms are designed based on GA, which is suitable for batch (or static) task allocation problems where the information of workers and tasks are known before running the algorithm. Therefore, the proposed algorithms cannot be applied to some MCS applications where the platform needs to make real-time decisions when receiving task requests.

3) *Incentive mechanism*: Task allocation and incentive design are two important research topics in MCS. Currently, research works on task allocation mainly focus on matching workers and tasks efficiently, and they assume that workers are willing to perform tasks given a simple incentive mechanism. Similarly, our model adopts a simple incentive mechanism. On the other hand, research works on incentive mechanism design [56], [57], [59] mainly focus on designing diverse thorough mechanisms to provide ordinary mobile users with sufficient incentives, and they use scenarios which do not require efficient task allocation methods. In the future work, we will investigate incorporating a more thorough incentive mechanism to the task allocation model.

7 Conclusion

This paper studies a multi-task allocation problem with time constraints in MCS. We first model the problem as a combinatorial optimization problem with two time constraints (i.e., the expected working time of workers and the valid time of tasks), which aims at maximizing the utility of the MCS platform. Then we prove that this problem is NP-complete and design two heuristic algorithms to solve the problem. Finally, the experiments based on synthetic and real-world datasets verify that the proposed algorithms outperform compared algorithms under different experiment settings.

Acknowledgments

This work was supported in part by the National Natural Science Foundations of China under Grant 61872149, in part by the Natural Science Foundations of Guangdong Province for Distinguished Young Scholar under Grant 2018B030306010, in part by the Guangdong Special Support Program under Grant 2017TQ04X482, in part by the Pearl River S&T Nova Program of Guangzhou under Grant 201806010088, and in part by the National Natural Science Foundations of China under Grant 61632013.

References

[1] 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, Nov. 2011.

[2] K. Chen, M. Lu, G. Tan, and J. Wu, “CRSM: Crowdsourcing based road surface monitoring,” in *Proc. IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2013, pp. 2151–2158.

[3] G. Alessandroni *et al.*, “SmartRoadSense: Collaborative road surface condition monitoring,” in *Proc. ACM Conf. Ubiquitous Comput.*, 2014, pp. 210–215.

[4] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, “Ear-phone: An end-to-end participatory urban noise mapping system,” in *Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2010, pp. 105–116.

[5] A. Goncalves, C. Silva, P. Morreale, and J. Bonafide, “Crowdsourcing for public safety,” in *Proc. IEEE Syst. Conf.*, 2014, pp. 50–56.

[6] M. N. K. Boulos *et al.*, “Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: Trends, OGC standards and application examples,” *Int. J. Health Geographics*, vol. 10, no. 1, 2011, Art. no. 67.

[7] W. Sun, Q. Li, and C.-K. Tham, “Wireless deployed and participatory sensing system for environmental monitoring,” in *Proc. IEEE Int. Conf. Sens. Commun. Netw.*, 2014, pp. 158–160.

[8] H. Alemdar and C. Ersoy, “Wireless sensor networks for healthcare: A survey,” *Comput. Netw.*, vol. 54, no. 15, pp. 2688–2710, 2010.

[9] S. He and K. G. Shin, “Steering crowdsourced signal map construction via Bayesian compressive sensing,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 1016–1024.

[10] C. Wu, Z. Yang, and Y. Liu, “Smartphones based crowdsourcing for indoor localization,” *IEEE Trans. Mobile Comput.*, vol. 14, no. 2, pp. 444–457, Feb. 2015.

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

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

[13] H. Xiong, D. Zhang, G. Chen, L. Wang, V. Gauthier, and L. E. Barnes, “iCrowd: Near-optimal task allocation for piggyback crowdsensing,” *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 2010–2022, Aug. 2016.

[14] S. Reddy, D. Estrin, and M. Srivastava, “Recruitment framework for participatory sensing data collections,” in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2010, pp. 138–155.

[15] M. H. Cheung, R. Southwell, F. Hou, and J. Huang, “Distributed time-sensitive task selection in mobile crowdsensing,” in *Proc. 16th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 157–166.

[16] X. Zhang, Z. Yang, and Y. Liu, “Vehicle-based bi-objective crowdsourcing,” *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 10, pp. 3420–3428, Oct. 2018.

[17] S. He, D.-H. Shin, J. Zhang, and J. Chen, “Toward optimal allocation of location dependent tasks in crowdsensing,” in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 745–753.

[18] Z. Song, C. H. Liu, J. Wu, J. Ma, and W. Wang, “QoI-aware multitask-oriented dynamic participant selection with budget constraints,” *IEEE Trans. Veh. Technol.*, vol. 63, no. 9, pp. 4618–4632, Nov. 2014.

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

[20] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, and D. Zhang, “TaskMe: Multi-task allocation in mobile crowd sensing,” in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 403–414.

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

[22] Mturk, Accessed: Jan. 20, 2018. [Online]. Available: https://www.mturk.com/

[23] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[24] P. Dutta *et al.*, “Common sense: Participatory urban sensing using a network of handheld air quality monitors,” in *Proc. 7th ACM Conf. Embedded Netw. Sensor Syst.*, 2009, pp. 349–350.

[25] K. Aberer *et al.*, "OpenSense: Open community driven sensing of environment," in *Proc. ACM SIGSPATIAL Int. Workshop Geo-Streaming*, 2010, pp. 39–42.
[26] S. Kim, C. Robson, T. Zimmerman, J. Pierce, and E. M. Haber, "Creek watch: Pairing usefulness and usability for successful citizen science," in *Proc. ACM SIGCHI Conf. Hum. Factors Comput. Syst.*, 2011, pp. 2125–2134.
[27] Accessed: Jan. 25, 2018. [Online]. Available: https://seeclickfix.com/
[28] Accessed: Jan. 25, 2018. [Online]. Available: https://www.weddar.com/
[29] N. Ramanathan *et al.*, "Ohmage: An open mobile system for activity and experience sampling," in *Proc. IEEE Int. Conf. Pervasive Comput. Technol. Healthcare*, 2012, pp. 203–204.
[30] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "PRISM: Platform for remote sensing using smartphones," in *Proc. 8th ACM Int. Conf. Mobile Syst. Appl. Services*, pp. 63–76.
[31] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowd-sensing-oriented mobile cyber-physical system," *IEEE Trans. Emerg. Topics Comput.*, vol. 1, no. 1, pp. 148–165, Jun. 2013.
[32] P. Cheng *et al.*, "Reliable diversity-based spatial crowdsourcing by moving workers," *Proc. VLDB Endowment*, vol. 8, no. 10, pp. 1022–1033, 2015.
[33] H. Li, T. Li, and Y. Wang, "Dynamic participant recruitment of mobile crowd sensing for heterogeneous sensing tasks," in *Proc. IEEE Int. Conf. Mobile Ad Hoc Sensor Syst.*, 2015, pp. 136–144.
[34] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *Proc. 21st ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2013, pp. 324–333.
[35] R. Estrada, R. Mizouni, H. Otrok, A. Ouali, and J. Bentahar, "A crowd-sensing framework for allocation of time-constrained and location-based tasks," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2725835.
[36] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.
[37] A. Hameed *et al.*, "A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems," *Computing*, vol. 98, no. 7, pp. 751–774, 2016.
[38] W. Gong, B. Zhang, and C. Li, "Task assignment in mobile crowd-sensing: Present and future directions," *IEEE Netw.*, vol. 32, no. 4, pp. 100–107, Jul./Aug. 2018.
[39] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong, "Task allocation in mobile crowd sensing: State-of-the-art and future opportunities," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3747–3757, Oct. 2018.
[40] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Oct.–Dec. 2017.
[41] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 84–106, 2013.
[42] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks*. Berlin, Germany: Springer, 2015, pp. 31–51.
[43] J. Bellingham, M. Tillerson, A. Richards, and J. P. How, "Multi-task allocation and path planning for cooperating UAVs," in *Cooperative Control: Models, Applications and Algorithms*. Berlin, Germany: Springer, 2003, pp. 23–41.
[44] H. Kellerer, U. Pferschy, and D. Pisinger, "Introduction to NP-completeness of knapsack problems," in *Knapsack Problems*. Berlin, Germany: Springer, 2004, pp. 483–493.
[45] E. Mezura-Montes and C. A. C. Coello, "Constraint-handling in nature-inspired numerical optimization: Past, present and future," *Swarm Evol. Comput.*, vol. 1, no. 4, pp. 173–194, 2011.
[46] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Methods Appl. Mechanics Eng.*, vol. 191, no. 11/12, pp. 1245–1287, 2002.
[47] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in *Proc. Int. Conf. Comput. Intell. Modelling Control Autom. and Int. Conf. Intell. Agents Web Technol. Internet Commerce*, 2005, pp. 1115–1121.
[48] J. R. Hoffman, M. S. Wilkes, F. C. Day, D. S. Bell, and J. K. Higa, "The roulette wheel: An aid to informed decision making," *PLoS Medicine*, vol. 3, no. 6, 2006, Art. no. e137.
[49] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 316–324.
[50] J. Yuan *et al.*, "T-drive: Driving directions based on taxi trajectories," in *Proc. 18th ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2010, pp. 99–108.
[51] Parking places of beijing, Accessed: Feb. 15, 2018. [Online]. Available: https://www.beijingcitylab.com/data-released-1/
[52] W. Jian-cheng, Z. Ya-qiao, Z. Xiao-juan, and R. Jian, "Floating car data based taxi operation characteristics analysis in beijing," in *Proc. WRI World Congress IEEE Comput. Sci. Inf. Eng.*, 2009, pp. 508–512.
[53] M. Dyer, N. Kayal, and J. Walker, "A branch and bound algorithm for solving the multiple-choice knapsack problem," *J. Comput. Appl. Math.*, vol. 11, no. 2, pp. 231–249, 1984.
[54] P. Somol, P. Pudil, and J. Kittler, "Fast branch & bound algorithms for optimal feature selection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 7, pp. 900–912, Jul. 2004.
[55] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. S. Shen, "Providing task allocation and secure deduplication for mobile crowdsensing via fog computing," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2018.2791432.
[56] X. Zhang, Z. Yang, W. Sun, Y. Liu, S. Tang, K. Xing, and X. Mao, "Incentives for mobile crowd sensing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 54–67, 2016.
[57] H. Cai, Y. Zhu, Z. Feng, H. Zhu, J. Yu, and J. Cao, "Truthful incentive mechanisms for mobile crowd sensing with dynamic smartphones," *Comput. Netw.*, vol. 141, pp. 1–16, 2018.
[58] X. Zhang, Z. Yang, Y. Liu, and S. Tang, "On reliable task assignment for spatial crowdsourcing," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 174–186, 2019.
[59] X. Zhang, L. Jiang, and X. Wang, "Incentive mechanisms for mobile crowdsensing with heterogeneous sensing costs," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3992–4002, 2019.

Xin Li received the BE degree in computer science and engineering from the South China University of Technology, Guangzhou, China, in 2018. He is currently working toward the master's degree at South China University of Technology, Guangzhou, China. His research interests include mobile crowdsensing and mobile computing.

Xinglin Zhang received the BE degree from the School of Software, Sun Yat-sen University, Guangzhou, China, in 2010, and the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, in 2014. He is currently an associate professor at the South China University of Technology, Guangzhou, China. His research interests include mobile crowdssensing, wireless ad-hoc/sensor networks, and mobile computing. He is a member of the IEEE and ACM.