


Article

Time-Efficient Allocation Mechanisms for Crowdsensing Tasks with Precedence Constraints [†]

Xiaocan Wu ¹, Yu-E Sun ^{2,3,*}, He Huang ^{1,3} , Yang Du ³ and Danlei Huang ¹

¹ The School of Computer Science and Technology, Soochow University, Suzhou 215006, China; 20184227017@stu.suda.edu.cn (X.W.); huangh@suda.edu.cn (H.H.); 20185227048@stu.suda.edu.cn (D.H.)

² The School of Rail Transportation, Soochow University, Suzhou 215131, China

³ The Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China; jannr@mail.ustc.edu.cn

* Correspondence: sunye12@suda.edu.cn

[†] This paper is an extended version of our paper published in Wu, X.; Huang, D.; Sun, Y.; Bu, X.; Xin, Y.; Huang, H. An Efficient Allocation Mechanism for Crowdsourcing Tasks with Minimum Execution Time. In Proceedings of the International Conference on Intelligent Computing 2017, Liverpool, UK, 7–10 August 2017.

Received: 19 April 2019; Accepted: 24 May 2019; Published: 29 May 2019



Abstract: Crowdsensing has emerged as an efficient and inexpensive way to perform specialized tasks by leveraging external crowds. In some crowdsensing systems, different tasks may have different requirements, and there may be precedence constraints among them, such as the Unmanned Aerial Vehicle (UAV) crowdsensing systems. Moreover, minimizing the total execution time is a regular target for finishing the crowdsensing tasks with precedence constraints. As far as we know, only a few existing studies consider the precedence constraints among crowdsensing tasks, and none of them can minimize the total execution time simultaneously. To tackle this challenge, an efficient allocation mechanism for tasks with precedence constraints is first proposed, which can minimize the total execution time. Then, a case study is given to show how to fit our mechanism in the UAV crowdsensing system. Finally, the simulation results show that the proposed mechanisms have good approximate optimal ratios under different parameter settings and are efficient for the UAV crowdsensing system as well.

Keywords: crowdsensing; task allocation; execution time minimization; precedence constraint; UAV

1. Introduction

With the emergence of various wireless technologies (4/5G, DSRC, and etc.), ubiquitous terminal equipment, such as smartphones, vehicles and UAVs, can collect real-time data from the environment and transmit the data to the IoT central server effectively [1–3]. As an important application of IoT, crowdsensing can leverage the power of large crowds to complete the complicated sensing tasks by using their smartphones or other mobile devices [4,5]. Compared with the conventional data collection methods, crowdsensing provides a low-cost and time-efficient solution for large-scale sensing tasks. With the dramatic proliferation of mobile devices, a set of crowdsensing systems have been implemented in recent year [6–17]. For instance, Kumar Rana et al. implemented an Ear-Phone system for monitoring the environmental noise pollution in urban areas through crowdsensing data collection [18].

Task allocation mechanism is crucial for crowdsensing, which directly decides the performance of the crowdsensing system. A variety of task allocation mechanisms have been proposed for crowdsensing systems [19–31]. For example, Reddy et al. proposed to maximize the spatial coverage

with limited resource [19]. Jaimes et al. designed a budget-constrained incentive mechanism for task allocation [20]. He et al. took travel time into consideration and proposed to maximize the spatial coverage [21]. Considering that crowdsensing tasks may have various requirements (such as the type of data, sensing periods, etc.) and workers have different skills and reliability levels. Li et al. proposed to dynamically select appropriate workers for given tasks while keeping the constraints satisfied [22], Jin et al. incorporated quality of data to design the incentive mechanisms for MCS systems [25]. Iijima et al. considered the individual preference in distributed environments and proposed an adaptive task allocation mechanism that maximizes the social utility [32].

However, all these studies assume that the tasks in the crowdsensing system can be performed simultaneously and ignore that the sensing tasks may have precedence constraints in some applications. A crowdsensing task with precedence constraints cannot be executed before its pre-order tasks are finished. Actually, many crowdsensing applications have multiple steps, which will cause the precedence constraints of sensing tasks. UAV crowdsensing system is a typical system with precedence constraints, where exist mainly six types of tasks: WASD (Wide Area Search and Destroy), ISR (Intelligence Surveillance and Reconnaissance), CAS (Close Air Support), SEAD (Suppression of Enemy Air Defense), AR (aerial refueling), and PS (precision strike) [33,34]. In this system, there exists an execution sequence among tasks. Another example is the MCS based traffic congestion monitoring system, which monitors the traffic condition through collecting the sensing data of vehicles on different major roads. When a traffic jam occurs, the system will publish tasks to find out the reason that causes this congestion, to monitor the progress of the events that cause traffic congestion or to verify the effectiveness of the traffic grooming strategy. Obviously, these tasks have precedence constraints, i.e., the system needs to find out the congestion reason before monitoring the progress of the events.

Designing an efficient task allocation mechanism for tasks with precedence constraints meets more challenge than the existing ones. First, time efficiency, i.e., the total execution time of all the tasks, is usually important for these crowdsensing systems. Generally, task requesters want the total execution time as short as possible, such as in the UAV system [35–38]. However, different tasks may have different requirements for users, and each user can only meet the requirements of some tasks. The platform can only allocate tasks to the user who meets their requirements. Since all the constraints are taken into account, it is a hard job to allocate the tasks to users optimally, especially for the case users arrive online. As far as we know, only a few studies [39–44] have considered the precedence constraints among different tasks. For example, Schwarzrock et al. proposed a task allocation mechanism for UAV system, which can increase the amount of performed tasks [45]. However, none of these studies can minimize the total execution time of all the tasks at the same time.

To address this challenge, the crowdsensing task allocation problem with precedence constraints is studied in this paper, and an efficient allocation mechanism with the goal of minimizing the total execution time of tasks is designed. The NP-hardness of the studied problem can be proved by reducing the problem studied in this work to a classic NP-hard problem of multiprocessor scheduling problem (the details are as shown in Section 2.3), which means the studied problem does not exist a polynomial-time algorithm to get the optimal solution. Therefore, a near-optimal allocation mechanism is proposed to solve it. The designed mechanism includes four steps, which are task level division, final task set construction, allocation priority sequence construction, and task allocation. In order to minimize the total execution time of tasks, the mechanism first divides the level of tasks based on their precedence constraints and computes expected finish time of each task by assuming that there are enough users for all the tasks. Since the total execution time of the tasks is bounded by the critical task t_c with the maximum expected finish time, the platform should first allocate a task to a user which can minimize the expected finish time of t_c . Based on this principle, the algorithm constructs an allocation priority sequence for tasks. When a user arrives, it greedily chooses the task with the highest priority in the allocation priority sequence to allocate until all the tasks have been finished. Then, a case study is given to show how to fit the proposed mechanism in the UAV systems by considering the features of the UAV system. Finally, the simulation results show that the proposed mechanisms are efficient for

crowdsensing systems with task precedence constraints. The main contributions of this work are listed as follows:

- An efficient task allocation algorithm for tasks with precedence constraints is designed. As far as we know, this is the first work which considers the precedence constraints of tasks and the proposed algorithm can minimize the total execution time of all the tasks.
- A case study is given to show how to fit the proposed mechanism in the UAV system, by considering the features of UAV task allocation problem.
- Extensive simulations are conducted to evaluate the performance of the proposed algorithm, and the results show that the proposed algorithm has good approximate optimal ratios under different parameter settings.

The remainder of the paper is organized as follows. The description of the system model is presented in Section 2. Then, the details of the proposed approximation algorithm are given in Section 3. Next, a case study is given to show how to fit the proposed mechanism in the UAV system in Section 4. Afterward, a variety of simulations are conducted to evaluate the mechanism in Section 5. Lastly, the conclusion of the whole work is presented in Section 6.

2. Preliminaries

In this section, the system model is first introduced in Section 2.1, and then, the formal formulation of the task allocation problem is in Section 2.2. After that, the NP-hardness of the studied problem is proved in Section 2.3.

2.1. System Model

The crowdsensing system studied in this work is shown in Figure 1, which including a crowdsensing platform, a task requester and a set of mobile device users $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$. At the beginning of the task allocation, the requester will submit a set of tasks, which is denoted as $\mathcal{T} = \{t_1, t_2, t_3, \dots, t_m\}$, to the crowdsensing platform. Each task $t_j \in \mathcal{T}$ can be presented as $t_j = \{C_j, h_j, l_j, D_j\}$, where C_j denotes the conditional task set of t_j , h_j is the expected performing time of task t_j , l_j is the location of task t_j , D_j is the description of t_j . Due to the precedence constraints among tasks, only when all the tasks in the conditional task set C_j have been finished, can the task t_j be assigned to a user to perform. However, if the conditional task set $C_j = \phi$ (i.e., the task t_j has no conditional task), the task t_j can be allocated by the platform at any time.

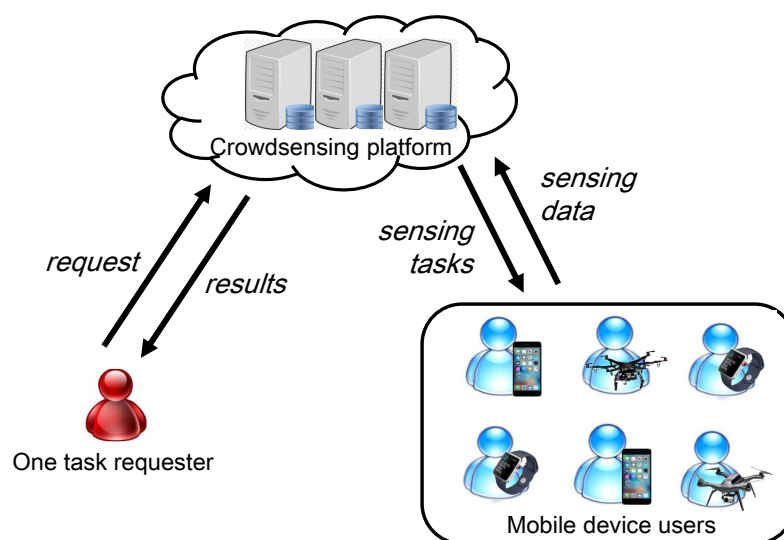


Figure 1. The structure of the crowdsensing system.

After receiving the task request from the requester, the platform will publish all the description of tasks to the users. The users arrive online, and each available user u_i will submit a set of interested tasks T_i to the platform. Based on the requirement of tasks and users, the platform will allocate the tasks to users one by one. Assume that each task only needs to be performed by one user, and the platform will allocate no more than one task to each user at each round. After finishing the allocated task, the platform will add the still available users to the waiting list, and treat it as a new arrival user. Use $u_i = \{T_i, PH_i\}$ to present user u_i , where PH_i is the expected execution time set of u_i , and each $ph_{i,j} \in PH_i$ is the expected execution time of u_i for performing task t_j .

In the UAV crowdsensing system, the task requester is the carrier, and each user is a UAV. The flight duration from the location of one task to another for different UAVs may be varied, which is an essential factor for the expected execution time of UAVs. Consider a UAV u_i has finished task t_j at the location l_j and is assigned to perform next task t_k at the location l_k . The travel duration of u_i can be presented as $td_{i,k} = TD(u_i, l_j, l_k)$, where $TD(u_i, l_j, l_k)$ is a function to calculate how long it generally takes for u_i to fly from l_j to l_k . Furthermore, if a UAV u_i is going to finish its first task t_k , the travel duration starts from its initial location l_i and it is denoted as $td_{i,k} = TD(u_i, \phi, l_k) = TD(u_i, l_i, l_k)$. Notice that h_k is the expected performing time of task t_k . Then, $ph_{i,k}$ is mainly decided by $td_{i,k}$ and h_k in the UAV system.

2.2. Problem Formulation

The goal of this work is to minimize the total execution time of the tasks with the precedence constraints.

For facilitate reading, we summarize some symbols that are used in this paper in Table 1.

Table 1. The descriptions of notations used in this paper.

Notation	Description
\mathcal{U}	a set of mobile device users
u_i	a mobile device user i
T_i	interested tasks submitted by user i
PH_i	the expected execution time set of user i
$ph_{i,j}$	the expected execution time of user i for performing task j
\mathcal{T}	the task set submitted by requester
t_j	a task j
C_j	the conditional task set of task j
h_j	the expected performing time of task j
l_j	the location of task j
D_j	the description of task j
$td_{i,k}$	the travel duration of user i to the location of task k
y_j	binary variable to represent whether task j is finished
a_j	binary variable to represent if task j is permitted to be allocated
$x_{i,j}$	binary variable to represent whether task j is allocated to user i
s_j	the earliest time that the platform can allocate the task j to a user
L_j	the level of task j
\mathcal{F}	the final task set
f_j	the order of task j in allocation priority sequence
h_j^e	the expected finishing time of task j
c_j^t	the recorded task with maximum expected finishing time in C_j

Let $y_j = \{0, 1\}$ represent whether the task t_j is finished. If the task t_j is a finished task, $y_j = 1$, otherwise, $y_j = 0$. Use $a_j = \{0, 1\}$ to denote if the task t_j is permitted to be allocated, and it has $a_j = \prod_{t_k \in C_j} y_k$. If task t_j satisfies the constraint $a_j = 1$, t_j can be allocated to a user. Further use $x_{i,j} = \{0, 1\}$ to indicate whether the platform allocates task t_j to user u_i . If t_j is assigned to the user u_i , $x_{i,j} = 1$, otherwise, $x_{i,j} = 0$. Suppose s_j is the expected beginning time of task t_j , (i.e.,

the earliest time that the platform can allocate task t_j to a user). Obviously, each s_j should satisfy that $s_j \geq \max_{t_k \in C_j} (s_k + \sum_{u_i \in \mathcal{U}} ph_{i,k} x_{i,k})$. Define the total time that the users finish the tasks in the task set as the execution time of a task set. Then, the goal of this work is to minimize the execution time of the task set \mathcal{T} .

Definition 1 (The Studied Task Allocation Problem(STAP)). *The studied crowdsensing task allocation problem can be defined as follows:*

$$\begin{aligned} \min \quad & \max_{t_j \in \mathcal{T}} (s_j + ph_{i,j} x_{i,j}) \\ \text{s.t.} \quad & s_j \geq \max_{t_k \in C_j} (s_k + \sum_{u_i \in \mathcal{U}} ph_{i,k} x_{i,k}) \\ & \dots \end{aligned}$$

The first constraint shows that the studied allocation mechanism should satisfy the precedence constraints of tasks. Therefore, t_j can be performed only when all the tasks in its conditional task set have been finished.

2.3. Analysis of the NP-Hardness

In the following, it will prove that the studied task allocation problem can be reduced to the multiprocessor scheduling problem, which is a well-known NP-hard problem [46]. The description of the multiprocessor scheduling problem is as follows: given a set of jobs and m processors, the goal of the multiprocessor scheduling problem is to find the minimum possible time required to schedule all jobs in the job set on m processors such that there is none overlap, where each job has a fixed processing time.

Theorem 1. *The studied task allocation problem (STAP) is NP-hard.*

Proof. Consider a simple case of the studied problem, where there is no precedence constraint among tasks, and each user is interested in all the tasks. Then, the task set in this problem can be viewed as the job set in the multiprocessor scheduling problem, and the users in the studied problem can be viewed as the processors in the multiprocessor scheduling problem. The performing time of tasks in this problem is equal to the processing time of jobs in the multiprocessor scheduling problem. Then, the goal of the problem is equivalent to find the minimum possible time required to schedule all jobs on the processors such that there is none overlap. As is known to all, the multiprocessor scheduling problem is NP-hard. Therefore, the problem studied in this work is also NP-hard, which finished the proof. \square

3. Algorithm Design

It has been proved that the studied task allocation problem (STAP) is NP-hard, which means an approximation mechanism with polynomial-time is demanded to solve it. The proposed mechanism includes four steps, which are task level division, final task set construction, allocation priority sequence construction, and task allocation. In the first step, the levels of tasks are divided based on their conditional task sets. Define a task that is not in any conditional task set of other tasks as a final task. The total execution time of all the tasks is determined by the finishing time of final tasks. Thus, the second step of the mechanism is the construction of the final task set. Next, the mechanism achieves its design goal by sorting the tasks in descending order based on their expected finishing time. Finally, the tasks are allocated to users according to the allocation priority sequence constructed in the third step.

3.1. Task Level Division

The level in the algorithm is used to denote the precedence constraints among tasks. In the studied model, tasks can be allocated to users only when all the tasks in their conditional task set have been finished. Thus, the level of all the tasks in C_j should be less than the level of t_j . Symbol L_j is used to denote the level of t_j . Obviously, each task t_j with $C_j = \phi$ is in the lowest level (i.e., $L_j = 1$). The details of how to divide the level of tasks are as shown in Algorithm 1.

Algorithm 1 task level division

Require:

the task set \mathcal{T}

Ensure:

```

 $\mathcal{L} = \{L_j\}_{t_j \in \mathcal{T}}$ 
1: Set  $\mathcal{T}' = \mathcal{T}$ ;
2: for each  $t_j$  in  $\mathcal{T}$  do
3:   Set  $C'_j = C_j$ ;
4:   Set  $k = 1$ ;
5:   while  $\mathcal{T}' \neq \phi$  do
6:     for each task  $t_j \in \mathcal{T}'$  do
7:       if  $C'_j = \phi$  then
8:         Set  $L_j = k$ ;
9:       for each task  $t_j \in \mathcal{T}'$  do
10:        for each task  $t_q \in C'_j$  do
11:          if  $L_q = k$  then
12:            Delete  $t_q$  from set  $C'_j$ ;
13:          if  $L_j = k$  then
14:            Delete  $t_j$  from set  $\mathcal{T}'$ ;
15:         $k++$ ;
16:   return  $\mathcal{L} = \{L_j\}_{t_j \in \mathcal{T}}$ ;

```

In Algorithm 1, it first makes a copy for each conditional task set C_j , which is denoted as C'_j . Initially the current task level $k = 1$. The task level division algorithm runs in an iterative way. In each iteration, it scans all the tasks in temporary task set \mathcal{T}' . When t_j is scanned, it will check whether the temporary conditional set $C'_j = \phi$ or not. If $C'_j = \phi$, set the task level of t_j equal to k (i.e., set $L_j = k$). After all the tasks in \mathcal{T}' have been scanned, it will delete the tasks with level k from the temporary conditional set of other tasks and delete t_j from the temporary task set \mathcal{T}' . Finally, the algorithm sets $k = k + 1$, and starts the next iteration until the task set $\mathcal{T}' = \phi$.

The following instance is given to express the algorithm more clearly. Suppose the task set $\mathcal{T} = \{t_1, t_2, t_3, \dots, t_6\}$ in Figure 2.

According to Algorithm 1, the level of each task in the task set \mathcal{T} is gotten. Apparently, tasks are divided into three levels. t_1, t_2, t_3 are in the first level, t_4, t_5 are in the second level and t_6 is in the last level.

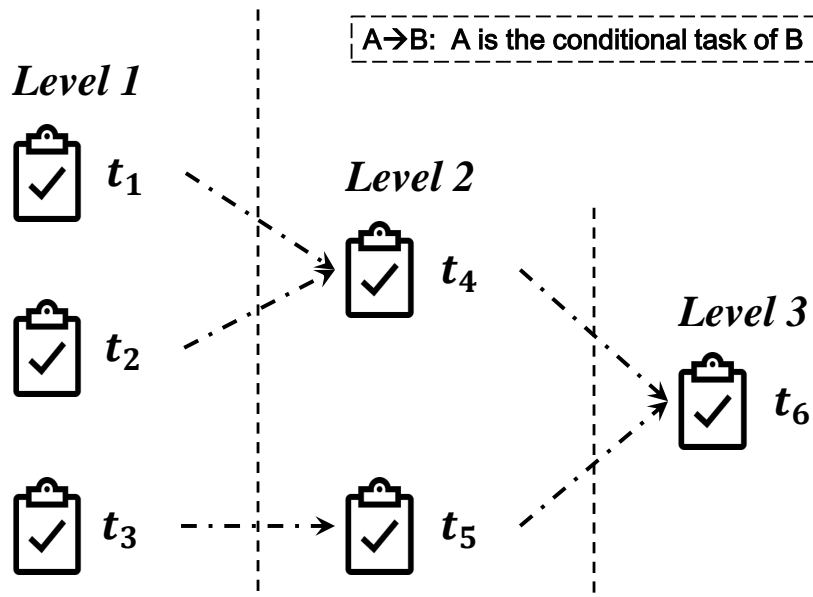


Figure 2. The instance for Algorithm 1.

3.2. Final Task Set Construction

Definition 2. Define the tasks that don't exist in any other tasks' conditional task set as final tasks. There exists at least one final task t_j in \mathcal{T} . If t_j is a final task, then $t_j \notin C_k$ stands for all the $t_k \in \mathcal{T}$.

Since the total execution time of the tasks is bounded by final tasks, construction of the final task set is performed before sorting the allocation priority of tasks. In Algorithm 2, all the tasks in \mathcal{T} are firstly assumed as final tasks. Then, they are checked one after another. When task t_j is checked, it will be deleted from the final task set \mathcal{F} if it exists in the conditional task set of at least one task. The details are as shown in Algorithm 2.

Algorithm 2 Final task set construction

Require:

the task set \mathcal{T}

Ensure:

the final task set \mathcal{F}

- 1: Set $\mathcal{F} = \mathcal{T}$;
 - 2: **for** each task $t_j \in \mathcal{F}$ **do**
 - 3: **for** each task $t_k \in \mathcal{T}$ **do**
 - 4: **for** each task $t_q \in C_k$ **do**
 - 5: **if** $t_j = t_q$ **then**
 - 6: Delete task t_j from the final task set \mathcal{F} ;
 - 7: **return** the final task set \mathcal{F} ;
-

Consider the instance in Section 3.1. There is no conditional task set C_k ($k \in [1, 6]$) in this example contains task t_6 . Thus, the constructed final task set $\mathcal{F} = \{t_6\}$. Obviously, all the tasks in \mathcal{T} should be done when all the final tasks in \mathcal{F} have been finished, and it is the feature of the final task.

3.3. Allocation Priority Sequence Construction

The optimization objective is bounded by the final tasks with the maximum expected finishing time. Thus, all the expected finishing time of the final tasks should be computed when tasks are

allocated to users. To achieve the designed goal, the algorithm sorts the tasks with their expected finishing time and constructs an allocation priority sequence. In the following, some important definitions are given first.

Definition 3. Task Sequence: A task sequence is sequence of tasks which satisfies the l -th task in the sequence should be in the conditional task set of the $l + 1$ -th task and this sequence ends in a final task.

Since the tasks can only be performed one by one in the task sequence, the expected finishing time of a task sequence is equal to the expected finishing time of the final task in the sequence.

Definition 4. The Critical Task Sequence: The critical task sequence is defined as the task sequence with maximum expected finishing time.

The allocation priority sequence construction algorithm runs in iterations. In each iteration, the critical task sequence of the task set \mathcal{T} has to be found first, then the algorithm puts the task with the lowest level in the critical task sequence into the allocation priority sequence. The details are as shown in Algorithm 3.

Suppose f_j is the order of task t_j in the allocation priority sequence. In each iteration, the algorithm aims to find a task sequence and its value is greater than any other task sequences. Note that the value of a task sequence is equal to the expected finishing time of the final task in the sequence. To get the expected finishing time of the final tasks, the expected finishing time of the tasks in their conditional task set should be calculated. Thus, Algorithm 3 first computes the expected finishing time of each task in \mathcal{T} .

Let h_j^e be the expected finishing time of task t_j . Note that the expected execution time of different users for the same task may be different. However, users arrive online in this work. Thus, it hardly to get the expected execution time of tasks before allocating. In order to solve this problem, the mechanism assumes the expected execution time of task t_j is h_j in this step. Then, it has $h_j^e = h_j$ when $L_j = 1$. When $L_j \geq 2$, $h_j^e = h_j + \max\{h_p^e\}_{t_p \in C_j}$. Notice that the algorithm computes the expected finishing time of tasks from low level to high level, and the levels of tasks in C_j are lower than t_j . Thus, $\{h_p^e\}_{t_p \in C_j}$ are known when it computes h_j^e . c_j^t is used to record the task with maximum expected finishing time among tasks in C_j , which can help to construct the critical task sequence.

After computing the expected finishing time of all the final tasks at the start of the algorithm, the mechanism begins to construct the priority sequence in iterations. As the expected finishing time of all the final tasks is calculated, the task t_q with maximum expected finishing time can be found, and construction of the critical task sequence of task t_q is available with the help of recorded c_j^t . Suppose t_p is the task with lowest level in the critical task sequence of final task t_q , t_p is put into the allocation priority sequence by setting $f_p = l$. Then, t_p is deleted from task set \mathcal{T} . If t_p is a final task, t_p should also be deleted from the final task set \mathcal{F} . Afterward, the algorithm finds tasks that are directly or indirectly related with the deleted t_p and computes their expected finishing time. Finally, the next iteration begins until $\mathcal{T} = \phi$.

Based on the information of tasks in Figure 2, the allocation priority sequence is gotten by continuously finding a new critical task sequence for a changed task set \mathcal{T} . Figure 3a is the situation of task set \mathcal{T} when the first task in allocation priority sequence has been found, and Figure 3b corresponds to the second task in the sequence. In Figure 3a, the task sequence $\langle t_3, t_5, t_6 \rangle$ is the critical task sequence, then, the task t_3 is the first task in allocation priority sequence. In Figure 3b, the task sequence $\langle t_2, t_4, t_6 \rangle$ is the critical task sequence in the changed task set $\mathcal{T} - \{t_3\}$, and the task t_2 is the second task in the allocation priority sequence. Furthermore, in order to find the third task in the allocation priority sequence, the mechanism ought to find the critical task sequence in the changed task set $\mathcal{T} - \{t_2, t_3\}$.

Algorithm 3 Allocation priority sequence construction**Require:**

the task set \mathcal{T} , the final task set \mathcal{F} ,

the refresh task sequence \mathcal{R} ;

Ensure:

the allocation priority sequence $\{f_j\}_{t_j \in \mathcal{T}}$;

- 1: **for** $k = 1$ to $\max\{L_j\}_{t_j \in \mathcal{T}}$ **do**
- 2: **for** each task in \mathcal{T} **do**
- 3: **if** $L_j = k$ **then**
- 4: Set c_j^k is the task with the maximum expected finishing time among all the tasks in C_j .
- 5: Compute the expected finishing time h_j^k ;
- 6: Set $l = 1$;
- 7: **while** $\mathcal{T} \neq \emptyset$ **do**
- 8: Find the final task $t_q \in \mathcal{F}$ with maximum expected finishing time;
- 9: Construct the critical task sequence of task t_q ;
- 10: Set $f_p = l$;
- 11: Find the task t_p with the lowest level in the critical task sequence of task t_q ;
- 12: Delete t_p from \mathcal{T} ;
- 13: **if** $t_p \in \mathcal{F}$ **then**
- 14: Delete t_p from \mathcal{F} ;
- 15: Gather all the rest tasks that have priority relationship with t_p in the sequence \mathcal{R} ;
- 16: **for** each task t_j in \mathcal{R} **do**
- 17: Set c_j^l is the task with the maximum expected finishing time among all the tasks in C_j .
- 18: Compute the expected finishing time h_j^l ;
- 19: Add all the rest tasks that have priority relationship with t_j to the back of sequence \mathcal{R} ;
- 20: Set $l = l + 1$;

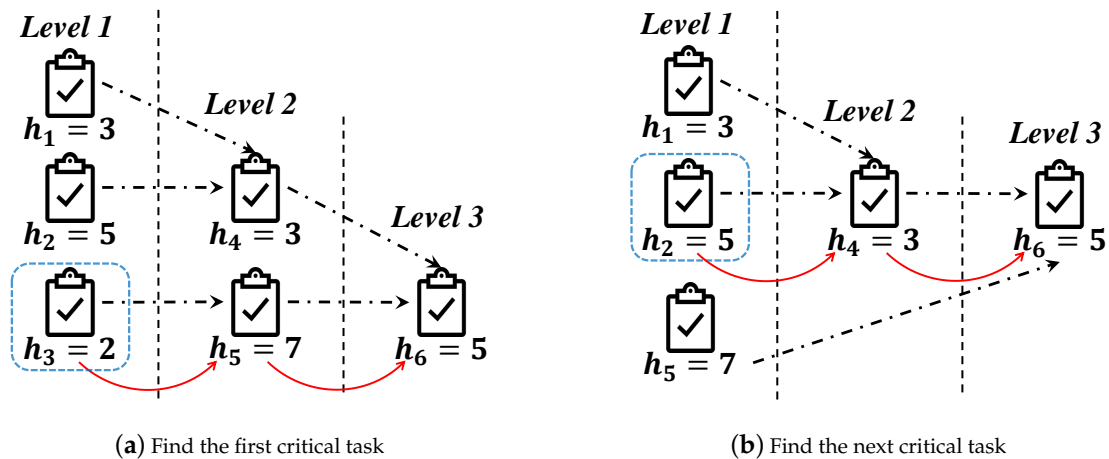


Figure 3. The instance for Algorithm 3.

3.4. Task Allocation

After constructing the allocation priority sequence, the platform allocates tasks to users according to the order in the constructed allocation priority sequence. The proposed task allocation mechanism runs iteratively. In each iteration, the platform greedily allocates one task to a user, which will minimize the expected total execution time of all tasks. The details are as follows:

Step 1: Sort the tasks in \mathcal{T} according to the allocation priority sequence.

Step 2: In each iteration, the task with the highest priority in the sorted task list should be found first, and its conditional task set is ϕ . Assume this task is t_j . Next, compute the expected execution time of all users interested in task t_j . Then, the user u_i who interested in t_j with the minimum $pt_{i,j}$ can be found. The platform allocates t_j to u_i in this iteration. After that, the platform deletes u_i from the available user set and deletes t_j from the task set \mathcal{T} . Then, the next iteration begins until the task set $\mathcal{T} = \phi$. In the case of there is no task can be allocated to users, and all the tasks and users remain in \mathcal{T} and \mathcal{W} should wait for new arrive users or some of the allocated tasks finished.

Step 3: When an allocated task t_j has finished by u_i , the platform deletes t_j from all the conditional task sets that include t_j . If u_i is still available, the platform adds u_i into the available user set, and views it as a new arrival user. Then, run step 2.

Note that the expected total execution time will be minimized if the tasks are performed in the order of the constructed allocation priority sequence. The proposed allocation mechanism greedily choose the task with the highest priority to allocate in each iteration, which means the mechanism can achieve a near-optimal total execution time.

4. A Case Study: Task Allocation Mechanism for UAV System

This section is to show how to fit the proposed task allocation mechanism in the UAV system.

Consider a Crowdsensing based UAV system, there exists a carrier, a control platform and a group of UAVs embedded with different kinds of sensing devices. At the beginning of each round allocation, the carrier first submits the tasks to the platform. The platform has an available UAV list, and each available UAV submits a set of tasks that it can perform to the platform. Then, the platform runs Algorithms 1 and 2 to compute the level of each task and construct the final task set.

As is introduced in Section 2.1, the expected execution time of a UAV for task t_j is mainly decided by the flight duration from the current location to L_j and the expected performing time of t_j in the UAV system. Although the flight time of different UAV may be varied, the expected performing time of different UAVs is similar for a fixed task. Therefore, the expected performing time of t_j can be assumed to equal to h_j for all the UAVs that have ability to perform t_j . In the step of constructing the allocation priority sequence, set $h_j^e = h_j + \max\{h_p^e\}_{t_p \in C_j}$. By running Algorithm 3, the platform can get the allocation priority sequence of tasks.

Then, the platform adds the UAVs in a waiting list, and allocates the tasks to them based on the allocation priority sequence. In each iteration of allocation, the platform allocates a task to a user that can minimize the total execution time of all the tasks, i.e., allocates a task with the lowest level in the critical sequence to the user with minimal expected execution time. Based on the proposed mechanism of constructing the allocation priority sequence, the task with the lowest level in the critical sequence is the task with the highest priority in the sequence. Assume t_j is this specific task. The expected execution time of a UAV u_i for performing t_j is $ph_{i,j} = td_{i,j} + h_j$.

When a UAV has finished an allocated task t_j , it will be added to the waiting list again. Moreover, the platform will delete t_j from all the conditional task sets that include t_j . This process goes on, until all the tasks have been finished.

5. Simulation

In this section, the settings of all the parameters are introduced first, and then extensive simulations are conducted to evaluate the proposed mechanisms.

5.1. Simulation Setting

A task is not always allocated immediately once it is available. More in details, when a task is available to be allocated, it might wait some time before being allocated. Furthermore, the total execution time of all the tasks is not likely to equal the theoretically optimal value of the allocation for the task set. Thus, the approximate optimal ratio is related to the performance of the algorithm in simulations.

Definition 5. The parameter Alg denotes the execution time of task set \mathcal{T} under the allocation of an algorithm, and Opt represents the theoretically optimal value of the allocation for the task set \mathcal{T} . Then, the approximate optimal ratio: $\eta = \frac{Alg}{Opt}$.

The setup of the simulation is as follows. In order to show the performance of the proposed algorithm, it varies the number of the released tasks, the number of the involved mobile device users and the level of the tasks set \mathcal{T} with the symbol of m, n, l . Besides, $m = |\mathcal{T}|$, $n = |\mathcal{U}|$ and $l = \max \{L_j\}_{t_j \in \mathcal{T}}$. The number of total tasks in each level is uniformly distributed in $[m/l - 5, m/l + 5]$. For any task t_j in task set \mathcal{T} , it has attributes of expected performing time h_j , conditional task set C_j , and its location l_j . The size of C_j is always distributed in $[1, 4]$ at random. In Figures 4–9, the parameter h_j appears to follow the uniform distribution in $U[20, 40]$. And in Figures 10 and 11 simulations, the parameter h_j can also be $U(30, 50)$ uniformly distributed. Each user u_i submits a set of tasks T_i that he is interested in performing, and the size of T_i is randomly generated in $[4, 10]$ or $[8, 14]$ in different experiments. Furthermore, in Figures 4, 5, 8–11, the parameter $|T_i|$ always follows $U[4, 10]$ uniform distribution. And in Figures 6 and 7 simulations, the parameter $|T_i|$ is also set as $U(8, 14)$ uniformly distributed. What’s more, to show that the proposed mechanism is available to be applied to the UAV system, settings about both tasks and users’ location are also made. In Figures 4–7, 10 and 11, both the locations of tasks and users are uniformly distributed in $U(0, 100)$. And in Figures 8 and 9, the location can also be normally distributed in $N(50, 3)$. If a user u_i is assigned to finish the task t_j , calculate the Euclidean distance of the user and task, and relate it to the travel duration of the user u_i for the task t_j . After that, the requested performing time of a user u_i to perform task t_j is determined.

In each case of $\langle m, n, l \rangle$, the simulation generates 2000 instances and takes the average value of them. The average value is the outcome of the case finally. The settings of all cases and the outcomes of simulations are shown in Tables 2 and 3.

Table 2. The description of different cases.

Case	Description				Fixed Parameter
	m	T _i	h _j	(l _j) _x and (l _j) _y	
A	200	U(4, 10)	U(20, 40)	U(0, 100)	l = 6 or n = 70
B	300	U(4, 10)	U(20, 40)	U(0, 100)	
C	400	U(4, 10)	U(20, 40)	U(0, 100)	
D	400	U(8, 14)	U(20, 40)	U(0, 100)	
E	300	U(4, 10)	U(30, 50)	U(0, 100)	
F	300	U(4, 10)	U(20, 40)	N(50, 3)	

Table 3. The approximate optimal ratios under different cases.

Case	Approximate Optimal Ratios												
	Number of Mobile Device Users (n)						Number of Levels (l)						
	50	60	70	80	90	100	110	6	8	10	12	14	16
A	1.413	1.316	1.236	1.209	1.183	1.163	1.117	1.234	1.160	1.133	1.116	1.094	1.084
B	1.679	1.521	1.500	1.387	1.362	1.292	1.234	1.511	1.414	1.332	1.267	1.211	1.165
C	1.999	1.801	1.708	1.610	1.503	1.429	1.357	1.702	1.541	1.431	1.331	1.286	1.252
D	1.802	1.715	1.529	1.455	1.344	1.293	1.249	1.607	1.425	1.293	1.251	1.223	1.204
E	1.743	1.620	1.520	1.453	1.381	1.308	1.246	1.553	1.432	1.352	1.305	1.235	1.201
F	1.586	1.462	1.380	1.351	1.262	1.238	1.172	1.387	1.292	1.217	1.185	1.151	1.127

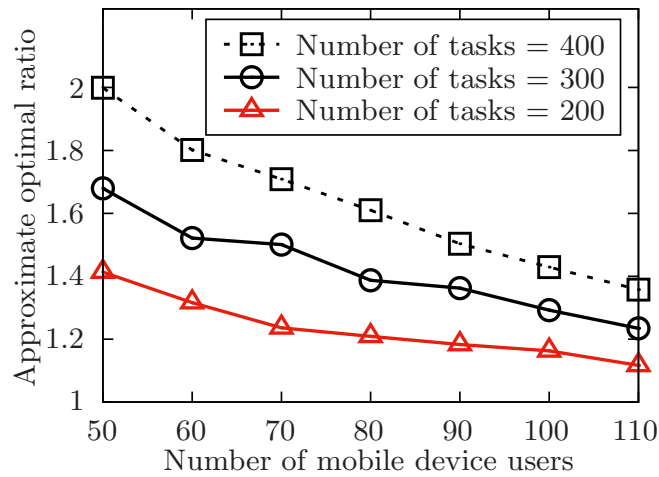


Figure 4. The approximate optimal ratio of the proposed algorithm vs. different n when $m = 200, 300, 400$.

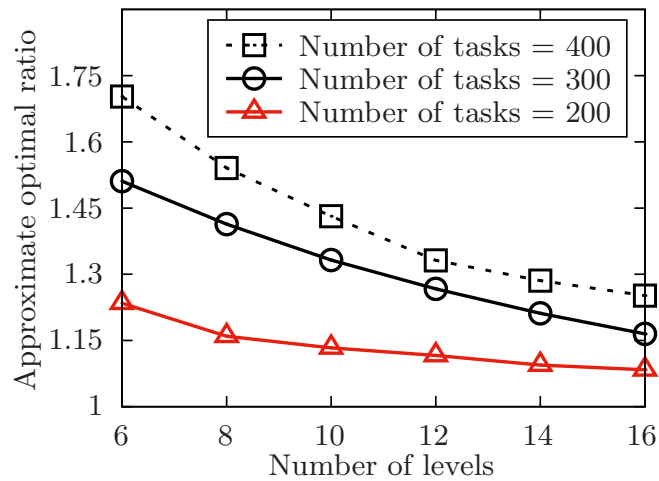


Figure 5. The approximate optimal ratio of the proposed algorithm vs. different l when $m = 200, 300, 400$.

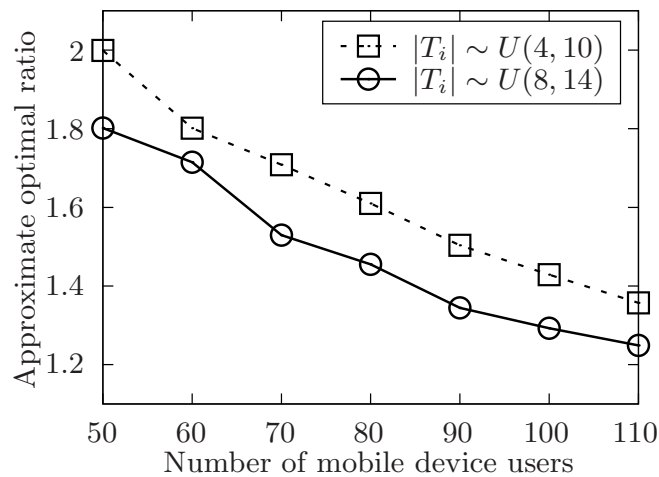


Figure 6. The approximate optimal ratio of the proposed algorithm vs. different n when $|T_i| \sim U(4, 10)$ or $U(8, 14)$.

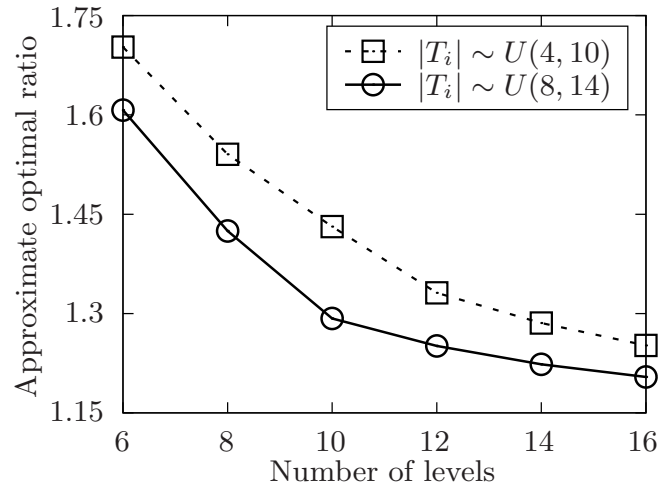


Figure 7. The approximate optimal ratio of the proposed algorithm vs. different l when $|T_i| \sim U(4, 10)$ or $U(8, 14)$.

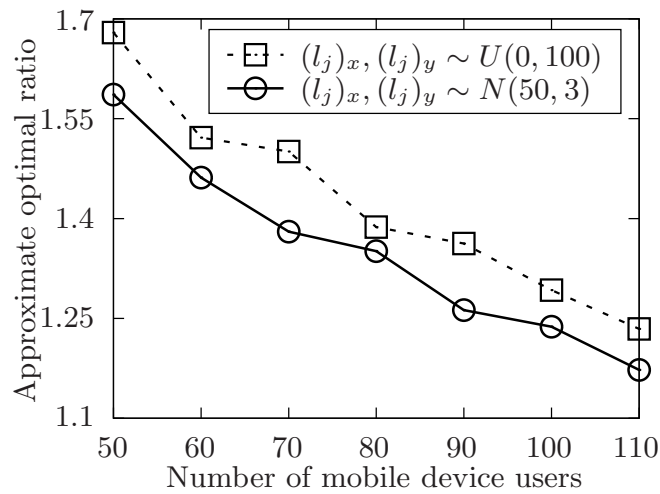


Figure 8. The approximate optimal ratio of the proposed algorithm vs. different n when $(l_j)_x, (l_j)_y \sim U(0, 100)$ or $N(50, 3)$.

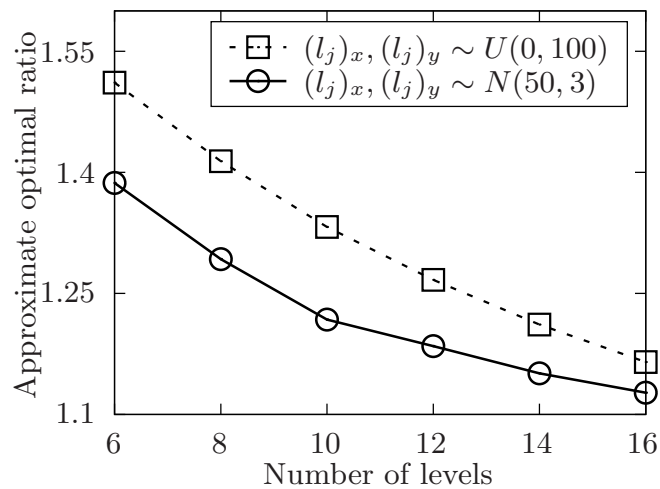


Figure 9. The approximate optimal ratio of the proposed algorithm vs. different l when $(l_j)_x, (l_j)_y \sim U(0, 100)$ or $N(50, 3)$.

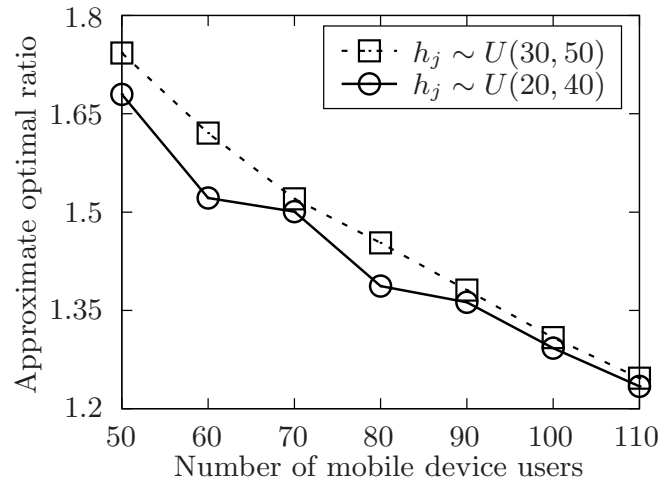


Figure 10. The approximate optimal ratio of the proposed algorithm vs. different n when $h_j \sim U(20, 40)$ or $U(30, 50)$.

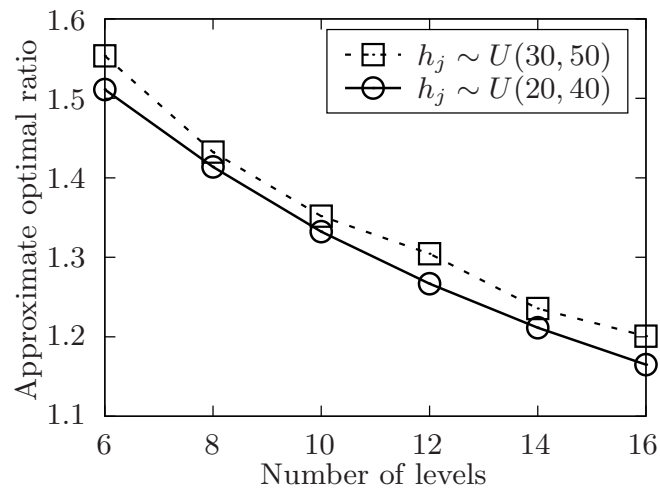


Figure 11. The approximate optimal ratio of the proposed algorithm vs. different l when $h_j \sim U(20, 40)$ or $U(30, 50)$.

5.2. Simulation Results

In Figures 4, 6, 8 and 10, the performance of the proposed algorithm is validated by changing the number of released tasks in different parameter settings. Regardless of other parameters' settings, it is obvious that when the number of involved users increases, the approximate optimal ratio decreases. As the theoretically optimal value of the allocation for the task set \mathcal{T} is only in connection with the structure of the task set \mathcal{T} , and the theoretically optimal value is fixed no matter how involved users change. When the number of involved users increases, the time of task's waiting to be allocated is likely to decrease, which would make the execution time of the task set \mathcal{T} decrease. Then, the approximate optimal ratio decreases. Therefore, the approximate optimal ratio decreases as the number of involved users increases.

In Figures 5, 7, 9 and 11, the performance of the proposed algorithm is shown by changing the level of the tasks set in different environment setting. Apparently, the approximate optimal ratio will decrease if the levels of the task set increases. It is because more levels of task set make the number of tasks in each level less, the tasks are more likely to be allocated once they are available, and the waiting time of tasks in task set \mathcal{T} is likely to decrease, which would make the execution time of the task set \mathcal{T} closer to the theoretically optimal value of the allocation for the task set. Thus, the approximate optimal ratio decreases as the level of task set increases.

In Figures 4 and 5, the performance of the algorithm is illustrated by changing the number of released tasks. In both two settings, the number of released tasks m ranges in $\{200, 300, 400\}$. In Figure 4, it changes the value of involved users n from 50 to 110 while the level of task set l is set to be 6. And in Figure 5, it changes the level of the task set l from 6 to 16 while the number of involved users n is set to be 70. Both two simulations show that the approximate optimal ratio increases with the increasing number of released tasks. As the theoretically optimal value of the allocation for the task set \mathcal{T} is only in connection with the structure of the task set \mathcal{T} , and the theoretically optimal value is fixed no matter how the number of released tasks changes. When the number of released tasks increases, the time of each task waiting to be allocated is likely to increase, and the execution time of the released task set is also to increase, which would increase the execution time of the task set \mathcal{T} .

In Figures 6 and 7, the size of involved users' interested task set T_i is in different range. In these two simulations, the performance of the algorithm is validated by changing the size of user's submitted task set. Furthermore, the size of user's submitted task set is uniformly distributed in $U(4, 10)$ or $U(8, 14)$. In both two settings, the number of released task m is fixed in 400. In Figure 6, the number of involved users n is changed from 50 to 110 while the level of task set l is set to be 6. And in Figure 7, it changes the level of task set l from 6 to 16 while the number of involved users n is set to be 70. Both two simulations show that the approximate optimal ratio decreases with the increasing number of user's submitted tasks. As the theoretically optimal value is fixed no matter how the number of released tasks changes. When the number of submitted tasks increases, and there exists some users in the available user list, the time of each task's waiting to be allocated is more likely to decrease, and the execution time of the released task set is also to decrease, which would make the execution time of the task set \mathcal{T} decrease.

In Figures 8 and 9, the performance of the algorithm is shown by changing the area of region that the released tasks and involved users locate in. In both two settings, the number of released task m is fixed to 300. In Figure 8, the number of involved users n ranges from 50 to 110 while the level of task set l is set to be 6. And in Figure 9, the level of task set l ranges from 6 to 16 while the number of involved users n is set to be 70. Both two simulations show that the approximate optimal ratio increases with the increasing of the area of regions. It is because when the area of region increases, the performing time of released tasks increases, then the time of each task waiting to be allocated is likely to increase, and the execution time of the released task set is also to increase, which would increase the execution time of the task set \mathcal{T} .

In Figures 10 and 11, the performance of the algorithm is illustrated by changing the expected performing time of released tasks. In both two settings, the number of released task m is fixed to 300. In Figure 10, the number of involved users n is changed from 50 to 110 while the level of task set l is set to be 6. And in Figure 11, it changes the level of task set l from 6 to 16 while the value of involved users n is set to be 70. Both two simulations show that the approximate optimal ratio increases with the increasing expected performing time of released tasks. When expected performing time of released tasks increases, the time users have to wait for each task to be allocated may increase, and the execution time of the released task set is also to increase, which would make the execution time of the task set \mathcal{T} increase.

6. Conclusions

In this paper, the precedence constraints of tasks are considered, and an efficient task allocation algorithm is designed for crowdsensing systems with the goal of minimizing the total execution time of the tasks. The proposed algorithm first divides tasks into multiple levels and finds all the final tasks. Then, it constructs an allocation priority sequence according to the expected finishing time of tasks, and allocates the tasks to users based on the constructed allocation priority sequence. Finally, a case study is given to show how to fit the designed mechanism in the UAV system. The simulation results verify the efficiency of the designed mechanism.

In future work, the deadlines of tasks and the available time intervals of users will be taken into consideration when designing an efficient task allocation mechanism for tasks with precedence constraints. Moreover, the plans for designing a mobile crowdsensing based traffic congestion monitoring system and exploring real-world experimentation for the proposed task allocation mechanism also deserve to be carried out.

Author Contributions: All the authors contributed to various degrees to ensure the quality of this work. Conceptualization, X.W. and Y.S.; Methodology, Y.S. and H.H.; Software, X.W., Y.D. and D.H.; Validation, Y.D. and D.H.; Formal analysis, X.W. and Y.S.; Investigation, Y.D.; Resources, Y.S. and H.H.; Data curation, X.W.; Writing—original draft preparation, X.W.; Writing—review and editing, X.W., Y.S. and H.H.; Visualization, D.H.; Supervision, Y.S.; Project administration, Y.S. and H.H.; Funding acquisition, Y.S. and H.H.

Funding: This research was partially funded by National Natural Science Foundation of China (NSFC) under Grant No. 61572342, No. 61672369, No. 61873177, and Natural Science Foundation of Jiangsu Province under Grant No. BK20161258.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Qiu, T.; Chen, N.; Li, K.; Atiquzzaman, M.; Zhao, W. How can heterogeneous Internet of Things build our future: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2011–2027. [[CrossRef](#)]
2. Qiu, T.; Liu, X.; Li, K.; Hu, Q.; Sangaiah, A.K.; Chen, N. Community-aware data propagation with small world feature for internet of vehicles. *IEEE Commun. Mag.* **2018**, *56*, 86–91. [[CrossRef](#)]
3. Qiu, T.; Zheng, K.; Han, M.; Chen, C.P.; Xu, M. A data-emergency-aware scheduling scheme for Internet of Things in smart cities. *IEEE Trans. Ind. Inform.* **2018**, *14*, 2042–2051. [[CrossRef](#)]
4. Ganti, R.K.; Ye, F.; Lei, H. Mobile crowdsensing: Current state and future challenges. *IEEE Commun. Mag.* **2011**, *49*, 32–39. [[CrossRef](#)]
5. Liu, J.; Shen, H.; Narman, H.S.; Chung, W.; Lin, Z. A Survey of Mobile Crowdsensing Techniques: A Critical Component for The Internet of Things. *ACM Trans. Cyber-Phys. Syst.* **2018**, *2*, 18. [[CrossRef](#)]
6. Leonardi, C.; Cappellotto, A.; Caraviello, M.; Lepri, B.; Antonelli, F. SecondNose: An Air Quality Mobile Crowdsensing System. In Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational, Helsinki, Finland, 26–30 October 2014; pp. 1051–1054.
7. Xu, C.; Li, S.; Zhang, Y.; Miluzzo, E.; Chen, Y. Crowdsensing the speaker count in the wild: Implications and applications. *IEEE Commun. Mag.* **2014**, *52*, 92–99. [[CrossRef](#)]
8. Wan, J.; Liu, J.; Shao, Z.; Vasilakos, A.; Imran, M.; Zhou, K. Mobile crowd sensing for traffic prediction in internet of vehicles. *Sensors* **2016**, *16*, 88. [[CrossRef](#)]
9. Pan, B.; Zheng, Y.; Wilkie, D.; Shahabi, C. Crowd Sensing of Traffic Anomalies Based on Human Mobility and Social Media. In Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Orlando, FL, USA, 5–8 November 2013; pp. 344–353.
10. Foremski, P.; Gorawski, M.; Grochla, K.; Polys, K. Energy-Efficient Crowdsensing of Human Mobility and Signal Levels in Cellular Networks. *Sensors* **2015**, *15*, 22060–22088. [[CrossRef](#)] [[PubMed](#)]
11. Rogstadius, J.; Vukovic, M.; Teixeira, C.A.; Kostakos, V.; Karapanos, E.; Laredo, J.A. CrisisTracker: Crowdsourced social media curation for disaster awareness. *IBM J. Res. Dev.* **2013**, *57*, 4:1–4:13. [[CrossRef](#)]
12. Adeel, U.; Yang, S.; McCann, J.A. Self-Optimizing Citizen-Centric Mobile Urban Sensing Systems. In Proceedings of the International Conference on Autonomic Computing, Philadelphia, PA, USA, 18–20 June 2014; USENIX Association: Berkeley, CA, USA, 2014; pp. 161–167.
13. Kumar, S.; Gil, S.; Katabi, D.; Rus, D. Accurate Indoor Localization with Zero Start-up Cost. In Proceedings of the Annual International Conference on Mobile Computing and Networking, Maui, HI, USA, 7–11 September 2014; pp. 483–494.
14. Yuan, N.J.; Zheng, Y.; Zhang, L.; Xie, X. T-Finder: A Recommender System for Finding Passengers and Vacant Taxis. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 2390–2403. [[CrossRef](#)]
15. Zheng, Y.; Liu, F.; Hsieh, H.P. U-Air: When Urban Air Quality Inference Meets Big Data. In Proceedings of the ACM SIGKDD, Chicago, IL, USA, 11–14 August 2013; pp. 1436–1444.
16. Zheng, Y.; Xie, X. Learning Travel Recommendations from User-generated GPS Traces. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 2:1–2:29. [[CrossRef](#)]

17. Du, Y.; Sun, Y.E.; Huang, H.; Huang, L.; Xu, H.; Bao, Y.; Guo, H. Bayesian Co-Clustering Truth Discovery for Mobile Crowd Sensing Systems. *IEEE Trans. Ind. Inform.* **2019**. [[CrossRef](#)]
18. Rana, R.K.; Chou, C.T.; Kanhere, S.S.; Bulusu, N.; Hu, W. Ear-phone: An End-to-end Participatory Urban Noise Mapping System. In Proceedings of the ACM/IEEE IPSN, Stockholm, Sweden, 12–16 April 2010; pp. 105–116.
19. Reddy, S.; Estrin, D.; Srivastava, M. Recruitment Framework for Participatory Sensing Data Collections. In *Pervasive Computing*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 138–155.
20. Jaimes, L.G.; Vergara-Laurens, I.; Labrador, M.A. A location-based incentive mechanism for participatory sensing systems with budget constraints. In Proceedings of the IEEE PerCom, Lugano, Switzerland, 19–23 March 2012; pp. 103–108.
21. He, S.; Shin, D.; Zhang, J.; Chen, J. Toward optimal allocation of location dependent tasks in crowdsensing. In Proceedings of the IEEE INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 745–753.
22. Li, H.; Li, T.; Wang, Y. Dynamic Participant Recruitment of Mobile Crowd Sensing for Heterogeneous Sensing Tasks. In Proceedings of the IEEE MASS, Dallas, TX, USA, 19–22 October 2015; pp. 136–144.
23. Boutsis, I.; Kalogeraki, V. On Task Assignment for Real-Time Reliable Crowdsourcing. In Proceedings of the IEEE ICDCS, Madrid, Spain, 30 June–3 July 2014; pp. 1–10.
24. Goel, G.; Nikzad, A.; Singla, A. Mechanism design for crowdsourcing markets with heterogeneous tasks. In Proceedings of the AAAI HCOMP, Pittsburgh, PA, USA, 2–4 November 2014; pp. 77–86.
25. Jin, H.; Su, L.; Chen, D.; Nahrstedt, K.; Xu, J. Quality of Information Aware Incentive Mechanisms for Mobile Crowd Sensing Systems. In Proceedings of the ACM MobiHoc, Hangzhou, China, 22–25 June 2015; pp. 167–176.
26. Feng, Z.; Zhu, Y.; Zhang, Q.; Ni, L.M.; Vasilakos, A.V. TRAC: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In Proceedings of the IEEE INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 1231–1239.
27. Sun, J. An incentive scheme based on heterogeneous belief values for crowd sensing in mobile social networks. In Proceedings of the IEEE GLOBECOM, Atlanta, GA, USA, 9–13 December 2013; pp. 1717–1722.
28. Ahmed, A.; Yasumoto, K.; Yamauchi, Y.; Ito, M. Distance and time based node selection for probabilistic coverage in People-Centric Sensing. In Proceedings of the IEEE SECON, Salt Lake City, UT, USA, 27–30 June 2011; pp. 134–142.
29. Zhou, Z.; Feng, J.; Gu, B.; Ai, B.; Mumtaz, S.; Rodriguez, J.; Guizani, M. When Mobile Crowd Sensing Meets UAV: Energy-Efficient Task Assignment and Route Planning. *IEEE Trans. Commun.* **2018**, *66*, 5526–5538. [[CrossRef](#)]
30. Liu, Y.; Guo, B.; Wang, Y.; Wu, W.; Yu, Z.; Zhang, D. TaskMe: Multi-task Allocation in Mobile Crowd Sensing. In Proceedings of the ACM UbiComp, Heidelberg, Germany, 12–16 September 2016; pp. 403–414.
31. Gong, W.; Zhang, B.; Li, C. Location-Based Online Task Scheduling in Mobile Crowdsensing. In Proceedings of the IEEE GLOBECOM, Singapore, 4–8 December 2017; pp. 1–6.
32. Iijima, N.; Sugiyama, A.; Hayano, M.; Sugawara, T. Adaptive Task Allocation Based on Social Utility and Individual Preference in Distributed Environments. *Procedia Comput. Sci.* **2017**, *112*, 91–98. [[CrossRef](#)]
33. Eun, Y.; Bang, H. Cooperative task assignment and path planning of multiple UAVs using genetic algorithm. In Proceedings of the AIAA Infotech at Aerospace 2007 Conference and Exhibit, Rohnert Park, CA, USA, 7–10 May 2007; p. 2982.
34. Boskovic, J.D.; Prasanth, R.; Mehra, R.K. A multi-layer autonomous intelligent control architecture for unmanned aerial vehicles. *J. Aerosp. Comput. Inf. Commun.* **2004**, *1*, 605–628. [[CrossRef](#)]
35. Schumacher, C.; Chandler, P.; Pachter, M.; Pachter, L. Constrained Optimization for UAV Task Assignment. In Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, RI, USA, 16–19 August 2004; p. 5352.
36. Bellingham, J.; Tillerson, M.; Richards, A.; How, J.P. Multi-Task Allocation and Path Planning for Cooperating UAVs. In *Cooperative Control: Models, Applications and Algorithms*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 23–41.
37. Leary, S.; Deittert, M.; Bookless, J. Constrained UAV mission planning: A comparison of approaches. In Proceedings of the IEEE ICCV Workshops, Barcelona, Spain, 6–13 November 2011; pp. 2002–2009.
38. Roberge, V.; Tarbouchi, M.; Labonté, G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. *IEEE Trans. Ind. Inform.* **2013**, *9*, 132–141. [[CrossRef](#)]

39. Chatterjee, A.; Borokhovich, M.; Varshney, L.R.; Vishwanath, S. Efficient and flexible crowdsourcing of specialized tasks with precedence constraints. In Proceedings of the IEEE INFOCOM, San Francisco, CA, USA, 10–15 April 2016; pp. 1–9.
40. Casbeer, D.W.; Holsapple, R.W. Column generation for a UAV assignment problem with precedence constraints. *Int. J. Robust Nonlinear Control* **2011**, *21*, 1421–1433. [[CrossRef](#)]
41. Peng, J.; Zhu, Y.; Zhao, Q.; Xue, G.; Zhu, H.; Cao, J.; Li, B. Fair Energy-Efficient Sensing Task Allocation in Participatory Sensing with Smartphones. *Comput. J.* **2017**, *60*, 850–865. [[CrossRef](#)]
42. Abououf, M.; Singh, S.; Otok, H.; Mizouni, R.; Ouali, A. Gale-Shapley Matching Game Selection—A Framework for User Satisfaction. *IEEE Access* **2019**, *7*, 3694–3703. [[CrossRef](#)]
43. Tang, C.; Wei, X.; Xiao, S.; Chen, W.; Fang, W.; Zhang, W.; Hao, M. A Mobile Cloud Based Scheduling Strategy for Industrial Internet of Things. *IEEE Access* **2018**, *6*, 7262–7275. [[CrossRef](#)]
44. de Moraes, R.S.; de Freitas, E.P. Experimental Analysis of Heuristic Solutions for the Moving Target Traveling Salesman Problem Applied to a Moving Targets Monitoring System. *Expert Syst. Appl.* **2019**. [[CrossRef](#)]
45. Schwarzrock, J.; Zacarias, I.; Bazzan, A.L.; de Araujo Fernandes, R.Q.; Moreira, L.H.; de Freitas, E.P. Solving task allocation problem in multi Unmanned Aerial Vehicles systems using Swarm intelligence. *Eng. Appl. Artif. Intell.* **2018**, *72*, 10–20. [[CrossRef](#)]
46. Garey, M.R.; Johnson, D.S. Complexity Results for Multiprocessor Scheduling Under Resource Constraints. In *Tutorial: Hard Real-time Systems*; IEEE Computer Society Press: Los Alamitos, CA, USA, 1989; pp. 205–219.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).