Multi-Task Assignment for CrowdSensing in Mobile Social Networks

Mingjun Xiao*, Jie Wu†, Liusheng Huang*, Yunsheng Wang‡, and Cong Liu§
*School of Computer Science and Technology / Suzhou Institute for Advanced Study,
University of Science and Technology of China, Hefei, P. R. China
†Department of Computer and Information Sciences, Temple University
‡Department of Computer Science, Kettering University
§School of Information Science and Technology, Sun Yat-sen University, Guangzhou, P. R. China

***Abstract*—Mobile crowdsensing is a new paradigm in which a crowd of mobile users exploit their carried smart devices to conduct complex computation and sensing tasks in mobile social networks (MSNs). In this paper, we focus on the task assignment problem in mobile crowdsensing. Unlike traditional task scheduling problems, the task assignment in mobile crowdsensing must follow the mobility model of users in MSNs. To solve this problem, we propose an oFfline Task Assignment (FTA) algorithm and an oNline Task Assignment (NTA) algorithm. Both FTA and NTA adopt a greedy task assignment strategy. Moreover, we prove that the FTA algorithm is an optimal offline task assignment algorithm, and give a competitive ratio of the NTA algorithm. In addition, we demonstrate the significant performance of our algorithms through extensive simulations, based on four real MSN traces and a synthetic MSN trace.**

***Index Terms*—Crowdsensing, delay tolerant network, mobile social network, task assignment**

I. Introduction

Mobile crowdsensing is a new paradigm involving a crowd of mobile users that exploit their carried smart devices to provide complex computation and sensing services in mobile social networks (MSNs) [9]. Since it can utilize the mobility of users to solve large-scale mobile sensing tasks, it has stimulated a number of attractive applications, such as urban WiFi characterization [7], traffic information mapping [6], and so on. Moreover, there have been some platforms, frameworks, and incentive mechanisms, designed for mobile crowdsensing in MSNs, such as in [3], [12], [16].

Consider that a mobile user in an MSN has some independent mobile sensing tasks, such as urban WiFi characterization, traffic information mapping, and so on. However, these tasks exceed its processing ability. Then, it requests other mobile users for help by starting a crowdsensing. In this crowdsensing, the requester will send tasks to and receive results from other mobile users, which generally involve large-size data transmissions. In order to decrease communication costs, mobile users prefer to adopt short-distance wireless communication technologies (e.g., WiFi or Bluetooth), instead of 3G/4G. Fig. 1 shows a simple example of this crowdsensing. The requester moves around, and sends tasks to another mobile user via Bluetooth when they encounter with each other, or via WiFi when they visit access points, respectively. After this mobile user processes the tasks, it sends the results back to the requester when they meet again. Then, an important problem is how the requester assigns these tasks to other mobile users, so as to minimize the average makespan of all tasks.

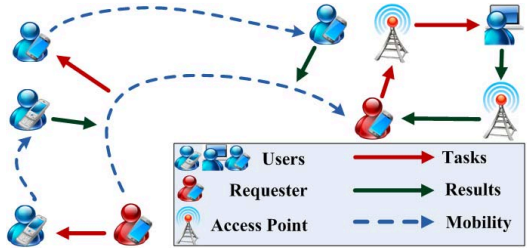


Fig. 1. Mobile crowdsensing in MSNs: the requester assigns tasks to and collects the results from other mobile users via Bluetooth when they meet, or via WiFi when they encounter access points, respectively.

In this paper, we focus on the above task assignment problem. In this problem, the makespan of a task not only includes the time of this task being conducted by a mobile user, but also contains the time of being sent from the requester to this user and the time for the result of this task being sent back to the requester. Nevertheless, the task can be assigned and the result can be returned, only when the requester meets the mobile user with some probability, which must follow the mobility model of users in MSNs. Such characteristics make our problem different from traditional parallel machine scheduling problems [1], [2], [4], [5]. The most related work to our problem is the serendipity system, in which a so-called Water Filling algorithm is proposed to allocate tasks among the mobile users in an MSN [15]. This work also takes the delivery time of the tasks and their results into consideration. However, it mainly focuses on minimizing the latest makespan of all tasks, so that the processing order of tasks is not important. In contrast, the tasks in our problem are independent. We only concern their average makespan.

To solve the above task assignment problem in mobile crowdsensing, we first propose a greedy oFfline Task Assignment (FTA) algorithm. The requester always assigns its tasks to the earliest idle mobile user, i.e., the user who has the minimum expected time to finish the tasks that have been assigned to it. What is more, unlike the Water Filling algorithm, the tasks in FTA must be assigned according to the ascending order of their workloads. In this way, the requester can achieve the optimal result in the offline case.

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

Further, by extending FTA, we propose an efficient oNline Task Assignment (NTA) algorithm, in which the requester repeatedly conducts the above greedy task assignment, when it encounters each mobile user, until all tasks are assigned. More specifically, our major contributions include:

1) We first introduce the task assignment problem into mobile crowdsensing in MSNs. Unlike traditional task scheduling problems, the requester not only needs to assign tasks, but also needs to recycle the results. Moreover, these can be conducted only when the requester meets other mobile users with some probabilities.
2) We propose an offline task assignment algorithm, i.e., FTA, in which a greedy task assignment strategy is adopted. Moreover, we prove that such a greedy strategy can achieve the minimum average makespan.
3) We also propose an online task assignment algorithm, i.e., NTA, by extending FTA. Furthermore, we prove that NTA can achieve a better performance than FTA. Additionally, we analyze the competitive ratio of NTA.
4) We conduct extensive simulations on four real traces and a synthetic trace to evaluate the FTA and NTA algorithms. The results show that the proposed algorithms can achieve smaller average makespans, compared with other algorithms.

The remainder of the paper is organized as follows. We introduce the network model, and the problem in Section II. The FTA and NTA algorithms are proposed in Sections III and IV, respectively. In Section V, we evaluate the performance of our algorithms through extensive simulations. After reviewing related work in Section VI, we conclude the paper in Section VII.

II. Model & Problem

A. Network Model

We consider an MSN that is composed of a crowd of mobile users, denoted by the set $V=\{v_0, v_1, ..., v_n\}$. Suppose that there is a user in this MSN, called the *requester*, who has some indivisible mobile sensing tasks. However, the total workload of these tasks is beyond its processing ability. Then, it starts crowdsensing. Other users in this MSN are assumed to be willing to participate in this crowdsensing due to some incentive mechanisms. Nevertheless, when these users conduct the crowdsensing tasks, they prefer to adopt the short-distance wireless communication model, so as to decrease the cost. More specifically, the requester moves around. If it encounters another mobile user, it assigns one or more tasks to this user. Then, this user will process these tasks. This might take some time. Also, the user will return the results of processed tasks to the requester, when they meet in the future.

Here, we say that two mobile users "*encounter*" or "*meet*", meaning that they move close and can directly communicate with each other via Bluetooth, or they enter the communication range of some access points respectively, so that they can indirectly contact each other via WiFi, as shown in Fig. 1. In this paper, we assume that the communication duration and bandwidth are enough for each user to receive tasks or return results. Moreover, we consider a widely-used mobility model [10], [17]. That is, the inter-meeting time between each user $v_i \in V$ and the requester follows the exponential distribution, whose rate parameter is λ_i. Moreover, we assume that the requester has known the rate parameter λ_i of each user v_i, which can be derived from its historical encounter records.

B. Problem

Consider a mobile crowdsensing in the above MSN. Without loss of generality, we let the requester be user v_0, and suppose that the requester has m indivisible mobile sensing tasks, denoted by $J=\{j_1, j_2, \cdots, j_m\}$. These tasks might be different types of tasks. Despite this, their workloads can be uniformly indicated by the sensing time, denoted as $\tau_1, \tau_2, \cdots, \tau_m$. For simplicity, we assume that all tasks in J need to be assigned to other users, and each task will be assigned to only one user.

In this paper, we focus on the average makespan of the tasks in J. The *makespan* of a task is the time that the requester finally receives the result of this task. We use $M(j)$ to denote the makespan of an arbitrary task $j \in J$. Moreover, we use a partition $\Pi=\{J_1, J_2, \cdots, J_n\}$ to denote a *task assignment strategy*, in which J_i $(1 \le i \le n)$ is an ordered set of tasks, satisfying $\sum_{i=1}^{n} J_i = J$ and $J_i \cap J_{i'} = \emptyset$ for $\forall J_i, J_{i'} \in \Pi$. If $J_i = \emptyset$, the requester will not assign any tasks to user v_i. Otherwise, the requester will send the tasks in J_i to user v_i, and then, user v_i will process these tasks according to their orders in J_i. For $\forall j, j' \in J_i$, we use $j \preceq j'$ to indicate that task j will be processed *prior* to j' in J_i (if $j \neq j'$). In addition, we use $AM(\Pi)$ to denote the *average makespan* of all tasks for a given task assignment strategy Π. That is,

$$AM(\Pi)=\frac{1}{m}\sum_{j\in J} M(j)|_{\Pi}. \quad (1)$$

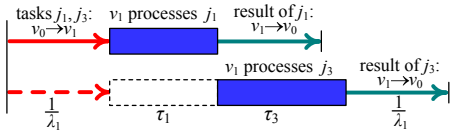
Then, our objective is to determine a task assignment strategy Π to minimize the average makespan $AM(\Pi)$. In the following sections, we consider two cases: the *offline task assignment* and the *online task assignment*. The former indicates that the requester makes the task assignment decision before it encounters any other mobile user, while the latter means that the requester dynamically makes the immediate task assignment decision at each time when it encounters a mobile user. To distinguish, we use Π_F and Π_N to denote the offline task assignment strategy and the online task assignment strategy, respectively. Moreover, we use Π_F^* and Π_{OPT} to denote the corresponding optimal strategies.

III. FTA: Offline Task Assignment

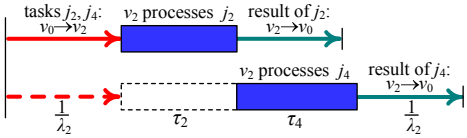
In this section, we first derive the formula to compute the average makespan for an arbitrary task assignment strategy, and then introduce the greedy task assignment strategy, followed by the detailed algorithm and the proof of optimality.

A. Basic Formula

Without loss of generality, we consider an arbitrary offline task assignment strategy $\Pi_F = \{J_1, J_2, \cdots, J_n\}$, and derive



(a) The expected makespans of tasks in $J_1 = \{j_1, j_3\}$: $M(j_1) = \frac{2}{\lambda_1} + \tau_1$; $M(j_3) = \frac{2}{\lambda_1} + \tau_1 + \tau_3$.



(b) The expected makespans of tasks in $J_2 = \{j_2, j_4\}$: $M(j_2) = \frac{2}{\lambda_2} + \tau_2$; $M(j_4) = \frac{2}{\lambda_2} + \tau_2 + \tau_4$.

Fig. 2. The average makespan of an offline task assignment strategy $\Pi_F = \{J_1, J_2\}$: $AM(\Pi_F) = \frac{1}{4}(M(j_1)+M(j_2)+M(j_3)+M(j_4))$.

the average makespan of all tasks for this task assignment strategy. More specifically, we have the following theorem:

Theorem 1: The average makespan $AM(\Pi_F)$ for the offline task assignment strategy $\Pi_F = \{J_1, J_2, \cdots, J_n\}$ satisfies:

$$AM(\Pi_F) = \frac{1}{m}\sum_{i=1}^{n}\sum_{j\in J_i}\Big(\frac{2}{\lambda_i} + \sum_{j'\in J_i \wedge j' \preceq j} \tau_{j'}\Big). \tag{2}$$

Proof: We consider an arbitrary task $j \in J_i$. It involves three phases. At the beginning, the task will be sent to user v_i by the requester. This happens only when the requester meets user v_i. Since the inter-meeting time of user v_i and the requester follows the exponential distribution with a rate parameter λ_i, the expected time of this phase is $\int_0^\infty \lambda_i t e^{-\lambda_i t} dt = \frac{1}{\lambda_i}$. In the second phase, user v_i will process the tasks in J_i. Note that user v_i will first process those tasks that are prior to j in J_i. Thus, the time of task j being processed in this phase is $\sum_{j'\in J_i \wedge j' \preceq j} \tau_{j'}$. In the third phase, the result of task j will be returned to the requester by user v_i when they have another meeting. The expected return time is still $\int_0^\infty \lambda_i t e^{-\lambda_i t} dt = \frac{1}{\lambda_i}$. Thus, the expected makespan of this task is $\frac{2}{\lambda_i} + \sum_{j'\in J_i \wedge j' \preceq j} \tau_{j'}$. By computing the average value of the expected makespans of all tasks in this way, we can get that the theorem holds. ■

Theorem 1 gives a formula, by which we can compute the average makespan for each possible offline task assignment strategy. According to the formula in this theorem, we can derive a basic property of the optimal offline task assignment strategy Π_F^*, as follows.

Theorem 2: Suppose the optimal offline task assignment strategy is $\Pi_F^* = \{J_1, J_2, \cdots, J_n\}$. Then, the tasks with small workloads will be processed first. More specifically, for $\forall j, j' \in J_i$ $(1 \le i \le n)$, if $\tau_j \le \tau_{j'}$, then the order of tasks j and j' in J_i satisfies $j \preceq j'$.

Proof: The contradiction method is adopted. Assume that there exists the tasks $j, j' \in J_i \in \Pi_F^*$, satisfying $\tau_j \le \tau_{j'}$ but $j' \preceq j$. Without loss of generality, we assume that j' and j are the k-th and h-th tasks in J_i, respectively, where $k < h$. Then, we construct another task assignment strategy Π_F' by exchanging the orders of the tasks j and j' in J_i. Computing $AM(\Pi_F^*)$ and $AM(\Pi_F')$ according to Eq. 2 in Theorem 1, and comparing them, we have:

$$AM(\Pi_F^*) - AM(\Pi_F') = \frac{1}{m}(h-k)\big(\tau_{j'} - \tau_j\big) > 0. \tag{3}$$

This means that the new task assignment strategy Π_F' can achieve a smaller average makespan than Π_F^*. This is a contradiction to the optimality of Π_F^*. Thus, the assumption is incorrect, and we have $j \preceq j'$. ■

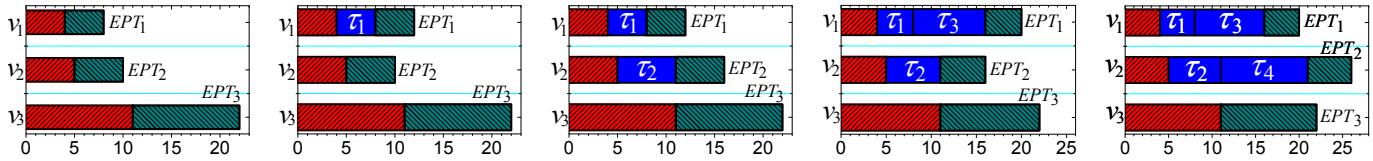
Fig. 2 shows a simple example of computing the average makespan for an offline task assignment strategy $\Pi_F = \{J_1, J_2\}$, where $J_1 = \{j_1, j_3\}$, and $J_2 = \{j_2, j_4\}$. The makespan of the task j_1 contains the expected time for the requester v_0 sending the task, the time for v_1 processing the task, and the expected time for v_1 returning the corresponding result. Thus, $M(j_1) = \frac{2}{\lambda_1} + \tau_1$. Besides, the makespan of the task j_3 needs to contain the waiting time for v_1 processing the task j_1 which is prior to j_3. That is to say, $M(j_3) = \frac{2}{\lambda_1} + \tau_1 + \tau_3$. Likewise, we can get $M(j_2) = \frac{2}{\lambda_2} + \tau_2$ and $M(j_4) = \frac{2}{\lambda_2} + \tau_2 + \tau_4$. Then, the average makespan for the task assignment strategy Π_F is $AM(\Pi_F) = \frac{1}{4}(M(j_1)+M(j_2)+M(j_3)+M(j_4))$. In this example, the workloads of the tasks j_1, j_2, j_3, and j_4 satisfy $\tau_1 < \tau_2 < \tau_3 < \tau_4$, and Π_F is actually the optimal task assignment strategy. In order to achieve the minimum average makespan, users v_1 and v_2 first process j_1 and j_2, respectively.

B. Basic Solution

According to Theorem 2, we adopt a greedy offline task assignment strategy by assigning the tasks with small wordloads first. Before the detailed solution, we first define a concept of expected processing time.

Definition 1 (Expected Processing Time): The expected processing time of a user, denoted by EPT, is the expected time for the user to meet the requester, process the tasks in hand, and return the result. More specifically, the expected processing time of a user is double the expected time of the user meeting the requester, plus the total workloads of the tasks in hand. For example, the expected processing time of a user v_i who has the tasks $\{j_{i_1}, j_{i_2}, \cdots, j_{i_k}\}$ is $EPT_i = \frac{2}{\lambda_i} + \tau_{i_1} + \tau_{i_2} + \cdots + \tau_{i_k}$. In addition, when the user has no tasks in hand, its expected processing time is double the expected time of the user meeting the requester.

The basic idea of FTA is that we always assign the minimum workload task among the tasks that have not been assigned to the user with the smallest expected processing time. At the beginning, we sort all tasks in the ascending order of their workloads. Without loss of generality, we assume that the workloads of the tasks satisfy $\tau_1 < \tau_2 < \cdots < \tau_m$. First, we assign the task j_1 to the user with the smallest expected processing time. At this time, the expected processing time of each user is double the expected time of the user meeting the requester. Without loss of generality, we assume that $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n$, which means that user v_1 is the user with the minimum expected processing time. Thus, the task j_1 is assigned to user v_1. Second, we focus on the task j_2. Now, user v_1 has the task j_1 in hand. Its expected processing time becomes $\frac{2}{\lambda_1} + \tau_1$. The minimum expected processing time will be $Min\{\frac{2}{\lambda_i} + \tau_1, \frac{2}{\lambda_2}, \cdots, \frac{2}{\lambda_n}\}$. Without loss of generality, we assume that user v_2 has the minimum expected processing



(a) The initial state (b) Assign the task j_1 (c) Assign the task j_2 (d) Assign the task j_3 (e) Assign the task j_4

Fig. 3. Example: greedily assign the tasks j_1, j_2, j_3, j_4 ($\tau_1 = 4, \tau_2 = 6, \tau_3 = 8, \tau_4 = 10$) to users v_1, v_2, v_3 ($\lambda_1 = 1/4, \lambda_2 = 1/5, \lambda_3 = 1/11$), and the optimal task assignment strategy is $\Pi_F^* = \{J_1, J_2, J_3\}$, where $J_1 = \{j_1, j_3\}$, $J_2 = \{j_2, j_4\}$, and $J_3 = \emptyset$.

time. Then, we assign the task j_2 to user v_2. In this way, the remaining tasks are assigned in turn.

Fig. 3 shows the process of greedily determining the optimal strategy Π_F^* through a simple example, in which four tasks j_1, j_2, j_3, j_4 are assigned to three users v_1, v_2, v_3. In this example, $\tau_1 = 4, \tau_2 = 6, \tau_3 = 8, \tau_4 = 10$, and $\lambda_1 = \frac{1}{4}, \lambda_2 = \frac{1}{5}, \lambda_3 = \frac{1}{11}$. At the beginning, the expected processing times of users v_1, v_2, v_3 are $\frac{2}{\lambda_1}$, $\frac{2}{\lambda_2}$, and $\frac{2}{\lambda_3}$, as shown in Fig. 3(a). Then, the task j_1 is assigned to the user v_1, as shown in Fig. 3(b). After this, the expected processing time of user v_1 becomes $\frac{2}{\lambda_1} + \tau_1$, which is larger than that of user v_2. Then, the task j_2 is assigned to the user v_2, as shown in Fig. 3(c). In the same way, the task j_3 is assigned to the user v_1 again in Fig. 3(d). Finally, the task j_4 is assigned to the user v_2 again in Fig. 3(e).

C. The Detailed Algorithm

Based on the above solution, we design the FTA algorithm, as shown in Algorithm 1. In Step 2, the set of tasks that are assigned to each user is initialized to be an empty set. Moreover, the expected processing time of each user is initialized in Step 3. Next, the algorithm assigns all tasks, in turn. In Step 5, the user with the minimum expected processing time is determined. Then, the task is assigned to this user in Step 6. After the assignment, the expected processing time of this user is updated in Step 7, for the next round of computation. The computation overhead is dominated by Step 5, which is $O(mn)$. In addition, as the input of this algorithm, the tasks are assumed to have been sorted in ascending order of their workloads in advance. This will lead to an extra computation overhead of $O(m \log m)$.

D. The Optimality

The FTA algorithm adopts an efficient greedy strategy to assign all tasks. Here, we prove that this greedy strategy is exactly optimal for the offline task assignment case.

Theorem 3: Suppose that the workloads of the tasks $j_1, j_2, \cdots, j_m$ satisfy $\tau_1 \le \tau_2 \le \cdots \le \tau_m$, among which the tasks $j_1, j_2, \cdots, j_{k-1}$ ($1 \le k \le m$) have been assigned according to the FTA algorithm. Denote the set of tasks that have been assigned to user v_i as J_i', and assume that the current expected processing time of this user is EPT_i. Then, the optimal task assignment strategy $\Pi_F^* = \{J_1, J_2, \cdots, J_n\}$ satisfies:

1) The user who currently has the minimum expected processing time will be assigned the maximum number of tasks in the following rounds of task assignments:

$$EPT_i = Min\{EPT_1, \cdots, EPT_n\} \Rightarrow |J_i - J_i'| = Max\{|J_1 - J_1'|, \cdots, |J_n - J_n'|\}, \quad (4)$$

Algorithm 1 The FTA Algorithm

Require: $J = \{j_1, j_2, \cdots, j_m : \tau_1 \le \tau_2 \le \cdots \le \tau_m\}$,
$V = \{v_1, v_2, \cdots, v_n : \lambda_1, \lambda_2, \cdots, \lambda_n\}$.
Ensure: $\Pi_F = \{J_1, J_2, \cdots, J_n\}$
1: **for** each user v_i **do**
2: $J_i = \emptyset$;
3: $EPT_i = \frac{2}{\lambda_i}$;
4: **for** task j from j_1 to j_m **do**
5: $i_{min} = argmin\{EPT_1, EPT_2, \cdots, EPT_n\}$;
6: Assign task j to $v_{i_{min}}$: $J_{i_{min}} = J_{i_{min}} + \{j\}$;
7: $EPT_{i_{min}} = EPT_{i_{min}} + \tau_{i_{min}}$;

where $|J_i - J_i'|$ is the number of tasks in the set $J_i - J_i'$.

2) The task j_k will be assigned to the user who currently has the minimum expected processing time:

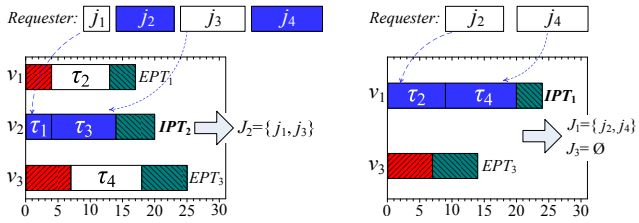
$$EPT_i = Min\{EPT_1, \cdots, EPT_n\} \Rightarrow j_k \in J_i. \quad (5)$$

Proof: 1) First, we prove part 1 by using the contradiction. Assume that although $EPT_i = Min\{EPT_1, \cdots, EPT_n\}$, $J_i - J_i'$ is not the set with the maximum tasks among $\{J_1 - J_1', \cdots, J_n - J_n'\}$. Without loss of generality, we let $|J_l - J_l'| = Max\{|J_1 - J_1'|, \cdots, |J_n - J_n'|\}$ ($1 \le l \le n, l \ne i$). Moreover, we assume that $J_i - J_i' = \{j_{i_1}, \cdots, j_{i_s}\}$, $J_l - J_l' = \{j_{l_1}, \cdots, j_{l_r}\}$, and $r > s$. Then, we construct another task assignment strategy $\Pi_F' = \{J_1, \cdots, \bar{J}_i, \cdots, \bar{J}_l, \cdots, J_n\}$, where $\bar{J}_i - J_i' = \{j_{l_1}, \cdots, j_{l_r}\}$, $\bar{J}_l - J_l' = \{j_{i_1}, \cdots, j_{i_s}\}$, by exchanging the tasks in $J_i - J_i$ and $J_l - J_l'$. After a careful computation on $AM(\Pi_F^*)$ and $AM(\Pi_F')$ according to Theorem 2 and Eq. 2 in Theorem 1, we have:

$$AM(\Pi_F^*) - AM(\Pi_F') = \frac{1}{m}(s - r)(EPT_i - EPT_l) > 0. \quad (6)$$

Thus, the new task assignment strategy Π_F' can achieve a smaller average makespan than Π_F^*. This is a contradiction to the optimality of Π_F^*. Due to the contradiction, we have that part 1 of the theorem holds.

2) We still adopt the contradiction method for part 2. Assume that $EPT_i = Min\{EPT_1, \cdots, EPT_n\}$, but $j_k \notin J_i$ (i.e., $j_k \notin J_i - J_i'$). Without loss of generality, we assume that task j_k is assigned to user v_l in the optimal offline task assignment strategy Π_F^*, i.e., $j_k \in J_l - J_l'$ ($l \ne i$), and assume that the task with the smallest workload in the task set $J_i - J_i'$ is j_h ($k < h \le m$). According to Theorem 2, tasks j_k and j_h are the first tasks to be processed in $J_l - J_l'$ and $J_i - J_i'$, respectively. Now, we construct another task assignment strategy Π_F' by exchanging the tasks $j_k \in J_l - J_l'$ and $j_h \in J_i - J_i'$. Then, computing $AM(\Pi_F^*)$ and $AM(\Pi_F')$ according to Eq. 2 in



(a) the requester determines J_2 when it meets v_2: $J_2=\{j_1,j_3\}$.

(b) the requester determines J_1 when it meets v_1: $J_1=\{j_2,j_4\}$.

Fig. 4. An example of online task assignment, in which the requester encounters users v_2, v_1, v_3, in turn.

Theorem 1, and comparing them, we have:

$$AM(\Pi_F^*)-AM(\Pi_F')=\frac{1}{m}(|J_l-J_l'|-|J_i-J_i'|)(\tau_k-\tau_h). \quad (7)$$

Since $EPT_i=Min\{EPT_1,\cdots,EPT_n\}$, we have $|J_i-J_i'|>|J_l-J_l'|$ according to the proof of part 1. Moreover, we have $\tau_k<\tau_h$ due to $k<h$. Thus, we can get

$$AM(\Pi_F^*)-AM(\Pi_F')>0. \quad (8)$$

This shows that the new task assignment strategy Π_F' can achieve a smaller average makespan than Π_F^*. This is a contradiction to the optimality of Π_F^*. Thus, the assumption is incorrect, and we have $j_k\in J_i-J_i'\subseteq J_i$. The theorem holds. ■

Theorem 3 shows that, in each round of task assignment, the current task with the smallest workload should always be assigned to the user who has the minimum expected processing time. This is exactly the task assignment strategy adopted by the FTA algorithm. Thus, we directly have:

Corollary 4: The FTA algorithm can achieve the optimal average makespan for the offline task assignment case.

IV. NTA: Online Task Assignment

In this section, we propose the online task assignment algorithm NTA, in which the task assignment decision is made only when the requester meets each mobile user. First, we introduce the basic solution, and then present the detailed algorithm, followed by the performance analysis.

A. The Basic Solution

Before the detailed solution, we first define a concept of instant processing time for each user, as follows.

Definition 2 (Instant Processing Time): The instant processing time, denoted by IPT, is the time for a mobile user, who has just encountered and received some tasks from the requester, to process these tasks and return the results. That is the total workloads of the tasks in hand plus the expected time for the user return the results to the requester. For example, the instant processing time of a user v_i who has the tasks $\{j_{i_1},j_{i_2},\cdots,j_{i_k}\}$ is $IPT_i=\frac{1}{\lambda_i}+\tau_{i_1}+\tau_{i_2}+\cdots+\tau_{i_k}$.

The basic idea of the NTA algorithm is that the requester makes the task assignment decisions only when it meets other mobile users, and the decisions are based on the instant processing time of the encountered users. The detailed solution is presented as follows. At the beginning, the requester holds all tasks in J. When it encounters a mobile user v_i, it starts an online task assignment process. More specifically, the requester first computes the instant processing time IPT_i of user v_i and the expected processing time of other users who have not been met by itself. Then, it adopts the same greedy strategy as that in FTA to assign tasks, while using the instant processing time IPT_i to replace the expected processing time EPT_i in FTA. That is, it always assigns the task with the minimum workload to the user who has the minimum instant processing time or expected processing time, until all tasks are assigned. Through this process, the requester will get a task assignment result $\{J_1,\cdots,J_i,\cdots,J_n\}$. However, only J_i among them is the final result. The requester only assigns the tasks in J_i to user v_i, while keeping the remaining tasks (i.e., the tasks in $J-J_i$) in hand. This ends a round of online task assignment. Actually, the requester might continue to meet other mobile users. For each encounter, it makes an online task assignment decision in the same way, to assign some tasks to the encountered user, until all tasks are assigned.

Fig. 4 shows an example, in which the requester has four tasks j_1,j_2,j_3,j_4 ($\tau_1=4,\tau_2=9,\tau_3=10,\tau_4=11$), and wishes to assign them to three mobile users v_1,v_2,v_3 ($\lambda_1=\frac{1}{4},\lambda_2=\frac{1}{6},\lambda_3=\frac{1}{7}$). In Fig. 4, the requester encounters v_2 first. Then, the requester computes the instant processing time of v_2 and the expected processing time of v_1 and v_3: $EPT_1=\frac{2}{\lambda_1}=8$, $IPT_2=\frac{1}{\lambda_2}=6$, and $EPT_3=\frac{2}{\lambda_3}=14$, among which IPT_2 is the smallest. Thus, the requester assigns the task j_1, which has the smallest workload, to user v_2. Then, IPT_2 becomes $IPT_2=\frac{1}{\lambda_2}+\tau_1=10$. Next, the requester adopts the same greedy strategy to assign the remaining tasks. As a result, we have $J_1=\{j_2\}$, $J_2=\{j_1,j_3\}$, and $J_3=\{j_4\}$, as shown in Fig. 4(a). Among them, only $J_2=\{j_1,j_3\}$ is the final result, and J_1, J_3 are temporary results. Then, the requester assigns the tasks j_1 and j_3 to v_2, while keeping the remaining tasks j_2 and j_4 in hand for future online task assignments. In the second step, the requester encounters user v_1. Then, the requester conducts the same task assignment process. The result is $J_1=\{j_2,j_4\}$, and $J_3=\emptyset$, as shown in Fig. 4(b). Moreover, $J_1=\{j_2,j_4\}$ is the final result. According to this result, the requester assigns the tasks j_2 and j_4 to v_1. Now, there are no remaining tasks. Then, $J_3=\emptyset$ is also the final result. Thus, the final result is $\Pi_N=\{J_1,J_2,J_3\}$, where $J_1=\{j_2,j_4\}$, $J_2=\{j_1,j_3\}$, and $J_3=\emptyset$.

B. The Detailed Algorithm

The detailed NTA algorithm is presented in Algorithm 2. When the requester meets an arbitrary user v_i, it first initializes the instant processing time IPT_i and the assigned task set J_i of the encountered user in Steps 1-2. In Steps 3-6, the expected processing time and the assigned task sets of other users are initialized. From Step 7 to Step 15, the requester determines whether each task in hand should be assigned to the encountered user v_i. First, the requester compares the instant processing time IPT_i and the expected processing time of other users in Step 8, so as to find the smallest one. The user who is related to this smallest value is denoted by $v_{i_{min}}$. If this user is exactly user v_i, the requester will assign the

Algorithm 2 The NTA Algorithm

Require: $J=\{j_1, j_2, \cdots, j_m : \tau_1 \leq \tau_2 \leq \cdots \leq \tau_m\}$,
$V=\{v_1, v_2, \cdots, v_n : \lambda_1, \lambda_2, \cdots, \lambda_n\}$.
Ensure: $\Pi_N=\{J_1, J_2, \cdots, J_n\}$
When the requester meets user v_i **do**
1: $IPT_i=\frac{1}{\lambda_i}$;
2: $J_i=\emptyset$;
3: $V=V-\{v_i\}$;
4: **for** each other user $v_l \in V$ **do**
5: $J_l=\emptyset$;
6: $EPT_l=\frac{2}{\lambda_l}$;
7: **for** task j from j_1 to j_m and $j \in J$ **do**
8: $i_{min}=argmin(\{IPT_i\}+\{EPT_l \mid v_l \in V\})$;
9: **if** $i_{min}=i$ **then**
10: Assign task j to v_i: $J_i=J_i+\{j\}$;
11: $IPT_i=IPT_i+\tau_i$;
12: $J=J-\{j\}$;
13: **else**
14: $J_{i_{min}}=J_{i_{min}}+\{j\}$;
15: $EPT_i=EPT_i+\tau_{i_{min}}$;
16: **return** J_i;

current task to this user, update the instant processing time IPT_i, and delete this task from J in Steps 10-12. Otherwise, the requester will temporarily assign this task to user $v_{i_{min}}$, and will update the expected processing time of this user in Steps 14-15. After this process, the requester will determine the tasks that should be assigned to the encountered user v_i. Then, the algorithm outputs the set of these tasks in Step 16. Next, when the requester meets another user, this algorithm will be conducted again to assign some of the remaining tasks to the new encountered user, and so on, until all tasks are assigned. The computation overhead is dominated by Step 8, which is $O(mn^2)$.

C. Performance Analysis

Compared to the offline FTA algorithm, the NTA algorithm adopts the same greedy strategy to make the decisions on task assignments, but the decisions are made only when the requester encounters other users, and the decisions are based on the instant processing time of the encountered users and the expected processing time of other users. By using the instant processing time, instead of the expected processing time of the encountered users, the requester can achieve a better performance. Here, we give the sketch proof of such a performance guarantee.

Theorem 5: The NTA algorithm can achieve a smaller average makespan than FTA, i.e., $AM(\Pi_N) \leq AM(\Pi_F^*)$.

Proof: Without loss of generality, we assume that the requester meets other users in the order of $v_1, v_2, \cdots, v_n$ during the crowdsensing. Moreover, we use Π_i to denote the temporary online task assignment result for the requester meeting user v_i. Now, we prove $AM(\Pi_0) \geq AM(\Pi_1) \geq \cdots \geq AM(\Pi_n)$, where Π_0=Π_F^* and Π_n=Π_N.

Consider the i-th online task assignment decision made by the requester when it meets user v_i ($1 \leq i \leq n$), and assume that the decision result is $\Pi_i = \{J_1, \cdots, J_{i-1}, J_i, J'_{i+1}, \cdots, J'_n\}$. In fact, $J_1, \cdots, J_{i-1}$ are the final results determined by the 1st, $\cdots$, $(i-1)$-th online task assignment decisions, respectively. At the i-th online decision, the requester assigns the remaining unassigned tasks, i.e. the tasks in $J - J_1 - \cdots - J_{i-1}$, and determines the final result J_i and the temporary results $J'_{i+1}, \cdots, J'_n$. This decision is made according to the greedy strategy adopted in the FTA algorithm. The only difference is that we use the instant processing time IPT_i to replace the expected processing time EPT_i. This is due to the reason that, when the requester has encountered user v_i, the time for the requester sending tasks to v_i changes from $\frac{1}{\lambda_i}$ (in EPT_i) to zero (in IPT_i). In Theorem 3, we have proven that such a greedy strategy can achieve the optimal (expected) average makespan. The optimality of this strategy still holds when we use IPT_i to replace EPT_i in the case that the requester meets user v_i. We can prove this by duplicating the proof of Theorem 3, while replacing EPT_i by IPT_i. To avoid the redundancy, we do not copy the detailed proof here. This result implies that $J_i, J'_{i+1}, \cdots, J'_n$ are the optimal task assignments until the i-th online decision. Thus, if we denote the result of the $(i-1)$-th online task assignment decision as $\Pi_{i-1} = \{J_1, \cdots, J_{i-1}, J''_i, J''_{i+1}, \cdots, J''_n\}$ (here, the final results $J_1, \cdots, J_{i-1}$ in Π_{i-1} are the same as those in Π_i, and $J''_i, J''_{i+1}, \cdots, J''_n$ are temporary results in this decision), then we have that the assignment $J_i, J'_{i+1}, \cdots, J'_n$ can achieve the smaller average makespan than $J''_i, J''_{i+1}, \cdots, J''_n$. This shows $AM(\Pi_{i-1}) \geq AM(\Pi_i)$. Due to the arbitrariness of i, we can get $AM(\Pi_0) \geq AM(\Pi_1) \geq \cdots \geq AM(\Pi_n)$ by the same analysis, implying the correctness of this theorem. ■

Further, we give the analysis on the competitive ratio of the NTA algorithm as follows.

Theorem 6: Assume that there is a god, who can foresee the mobilities of all mobile users as to know at what time the requester will meet which user. Based on this, the god can give an ideal optimal online task assignment strategy, denoted by Π_{OPT}. Then, we have:

1) The average makespan for the task assignment strategy Π_N produced by NTA satisfies:
$$AM(\Pi_N) - AM(\Pi_{OPT}) \leq \sum_{i=1}^{n} \frac{2}{\lambda_i}. \quad (9)$$
2) The competitive ratio of the NTA algorithm satisfies:
$$\frac{AM(\Pi_N)}{AM(\Pi_{OPT})} \leq 1 + \frac{m \sum_{i=1}^{n} \frac{2}{\lambda_i}}{\sum_{j=1}^{m} \tau_j}. \quad (10)$$

Proof: Without loss of generality, we assume that the requester meets users $v_1, v_2, \cdots, v_n$ at the time $t_1, t_2, \cdots, t_n$, and it also takes the time $t'_1, t'_2, \cdots, t'_n$ for these users to return their results to the requester, respectively. Moreover, we assume that the ideal optimal solution given by the god is $\Pi_{OPT} = \{J_1^*, \cdots, J_n^*\}$. Since the god has known the time at which the requester meets other users, and the time of other users returning results, this ideal optimal task assignment can

TABLE I
STATISTICS OF THE REAL TRACES.

Trace	Contacts	Length (hours)	Requester	Other users
Intel	2,766	99.8	9	128
Cambridge	6,732	145.6	12	223
Infocom	28,216	76.6	41	264
UMassDieselNet	227,657	95.3	4	36

be seen as a special offline task assignment. Then, we can use Eq. 2 in Theorem 1 to calculate the average makespan:

$$AM(\Pi_{OPT})=\frac{1}{m}\sum_{i=1}^{n}\left((t_i+t'_i)|J_i^*|+\sum_{j\in J_i^*}\sum_{j'\in J_i^*\wedge j'\preceq j}\tau_{j'}\right). \quad (11)$$

On the other hand, we consider an offline task assignment case without a god. Although there is no god, the requester still uses Π_{OPT} as its real offline task assignment strategy, i.e., $\Pi_F=\Pi_{OPT}$. Then, according to Theorem 1, we have:

$$\begin{aligned}AM(\Pi_F)&=\frac{1}{m}\sum_{i=1}^{n}\left(\frac{2}{\lambda_i}|J_i^*|+\sum_{j\in J_i^*}\sum_{j'\in J_i^*\wedge j'\preceq j}\tau_{j'}\right)\\&=AM(\Pi_{OPT})+\frac{1}{m}\sum_{i=1}^{n}\left(\frac{2}{\lambda_i}-t_i-t'_i\right)|J_i^*|. \quad (12)\end{aligned}$$

Note that, as an offline task assignment solution without a god, Π_F is not optimal. Actually, we have $AM(\Pi_F)\geq AM(\Pi_F^*)$ according to Corollary 4. On the other hand, according to Theorem 6, we have $AM(\Pi_F^*)\geq AM(\Pi_N)$. Thus, we can get $AM(\Pi_F)\geq AM(\Pi_N)$. Replacing $AM(\Pi_F)$ in this inequality by Eq. 12, we have:

$$\begin{aligned}AM(\Pi_N)-AM(\Pi_{OPT})&\leq\frac{1}{m}\sum_{i=1}^{n}\left(\frac{2}{\lambda_i}-t_i-t'_i\right)|J_i^*|\\&\leq\frac{1}{m}\sum_{i=1}^{n}\frac{2m}{\lambda_i}=\sum_{i=1}^{n}\frac{2}{\lambda_i}. \quad (13)\end{aligned}$$

Thus, the part 1 of this theorem is correct. Further, we can get $AM(\Pi_{OPT})\geq\frac{1}{m}\sum_{j=1}^{m}\tau_j$ according to Eq. 11. Therefore, according to Eq. 13, we have:

$$\frac{AM(\Pi_N)}{AM(\Pi_{OPT})}\leq 1+\frac{\sum_{i=1}^{n}\frac{2}{\lambda_i}}{AM(\Pi_{OPT})}\leq 1+\frac{m\sum_{i=1}^{n}\frac{2}{\lambda_i}}{\sum_{j=1}^{m}\tau_j}. \quad (14)$$

Thus, this theorem holds. ■

Theorem 6 shows that the absolute error of NTA is no more than a fixed value, i.e., $\sum_{i=1}^{n}\frac{2}{\lambda_i}$, which only depends on the expected meeting time between the requester and other users. When the average expected meeting time is very small, our algorithm can even achieve the nearly optimal result. On the other hand, the competitive ratio of NTA is subject to both the average workload of tasks and the expected meeting time. When the average workload is very large, the competitive ratio of NTA will be very close to 1, although the absolute error might change not much. In fact, our simulation results in the next section have also captured these observations.

V. EVALUATION

We conduct extensive simulations to evaluate the performances of the proposed algorithms. The compared algorithms, the traces that we used, the simulation settings, and the results are presented as follows.

A. Algorithms in Comparison

In order to evaluate the performances of our algorithms, we implement the Water Filling (*WF*) algorithm [15] and the ideal optimal online task assignment (*OPT*) algorithm, discussed in Section IV. Besides, according to classic parallel machine scheduling mechanisms, we also design and implement another algorithm, denoted by *LF* (Largest-First) [2].

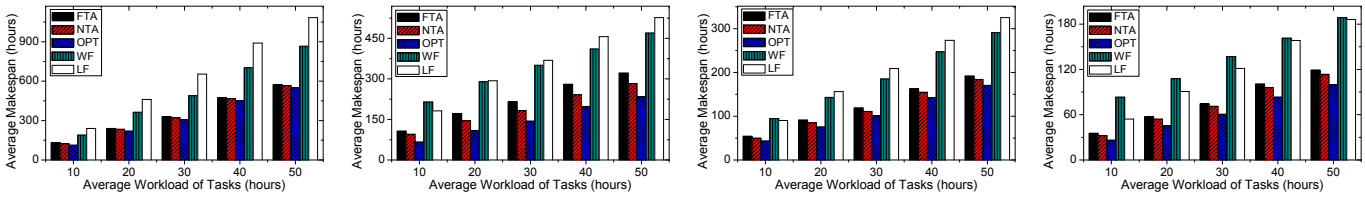
More specifically, the WF algorithm assigns the tasks of the requester, in turn, to the earliest idle user, i.e., the user who has the minimum expected processing time [15]. Different from FTA and NTA, the WF algorithm assigns the tasks according to their initial orders, i.e., $j_1, j_2, \cdots, j_m$. The OPT algorithm just makes the optimal offline task assignment decision according to the real time when the requester meets other users. Like FTA, NTA, and WF, the LF algorithm also assigns the tasks to the earliest idle users, but these tasks are assigned according to the descending order of their workloads, which is exactly the reverse order of that in FTA.

B. Real-traces Used and Simulation Settings

The *Cambridge Haggle Trace* [14] includes three traces of Bluetooth device connections by people carrying mobile devices (iMotes) over a certain number of days. These traces are collected by different groups of people in office environments, conference environments, and city environments, respectively. The nodes in the trace are classified into two groups: internal nodes and external nodes. Since there is no meeting record between internal nodes, we only use these internal nodes as requesters, and let the external nodes receive and process tasks. Table I shows some statistics of the traces that we used.

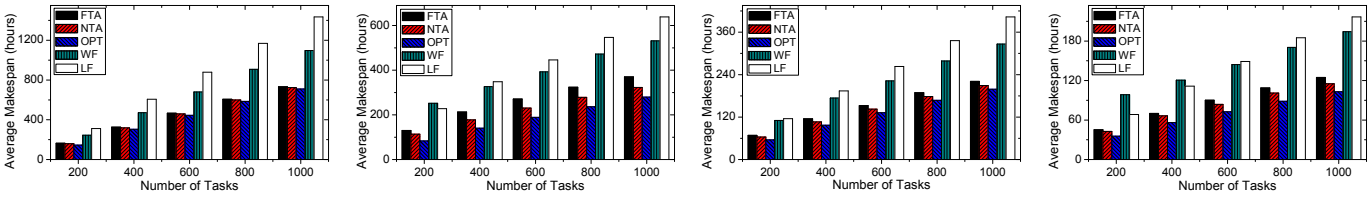
The *UMassDieselNet Trace* [13] contains the bus-to-bus contacts (the durations of which are relatively short) of 40 buses. Our simulations are performed on traces collected over 55 days during the Spring 2006 semester, with weekends, Spring break, and holidays removed due to reduced schedules. The bus system serves approximately ten routes. There are multiple shifts serving each of these routes. Shifts are further divided into morning (AM), midday (MID), afternoon (PM), and evening (EVE) sub-shifts. Drivers choose buses at random to run the AM sub-shifts. At the end of the AM sub-shift, the bus is often handed over to another driver to operate the next sub-shift on the same route, or on another route. For this trace, we select 4 buses with long trace records as the requesters, and let the remaining buses act as other mobile users.

In addition, we estimate the rate parameter λ_i of each user v_i by using the ratio of its meeting times with requesters and the total duration in the traces. Moreover, we randomly produce the tasks for each requester. The number of tasks is selected from $\{200, 400, \cdots, 1000\}$. The average workload of all tasks, denoted by τ, is selected from $\{10, 20, \cdots, 50\}$



(a) Cambridge Haggle: Intel (b) Cambridge Haggle: Cambridge (c) Cambridge Haggle: Infocom (d) UMassDieselNet

Fig. 5. Performance comparisons on real traces: the average makespan vs. the average workload of tasks (The number of tasks $m = 300$).



(a) Cambridge Haggle: Intel (b) Cambridge Haggle: Cambridge (c) Cambridge Haggle: Infocom (d) UMassDieselNet

Fig. 6. Performance comparisons on real traces: the average makespan vs. the number of tasks (The average workload of tasks $\tau = 20$).

(hours). Moreover, the maximum variance of the workload of each task is also τ. That is, the workload of each task is randomly selected from $[0, 2\tau]$.

C. Synthetic Traces and Simulation Settings

In order to evaluate the performances of our algorithms with different numbers of users and inter-meeting times, we also conduct a series of simulations on synthetic traces. First, we determine *the number of mobile users*, which is selected from $\{100, 200, \cdots, 1000\}$. Then, we randomly select $5\% \sim 10\%$ of these users as the requesters. Next, we determine *the average rate parameter* λ for the exponentially distributed inter-meeting time between the requesters and other users, which is selected from $\{0.01, 0.02, \cdots, 0.10\}$ (1/hour). Then, for each pair of requesters and mobile users, we randomly select a value from $[0, 2\lambda]$ as the rate parameter between them. Based on these network parameters, we construct an MSN to produce the synthetic trace. Finally, like the simulations on the real traces, we also generate the tasks for each requester, where the number of tasks is selected from $\{100, 200, \cdots, 1000\}$, and the average workload is selected from $\{5, 10, \cdots, 50\}$ (hours).

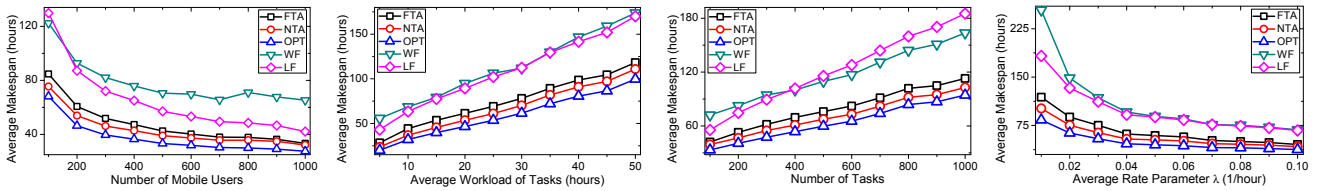
D. Evaluation Results

First, we evaluate the performances of the five algorithms through two groups of simulations on real traces, with different numbers of tasks and diverse average workloads. In the first group of simulations, we conduct these algorithms by changing the average workload, while keeping the number of tasks fixed. The results of the average makespans are shown in Fig. 5. For each real trace, the average makespans of FTA and NTA are more close to that of OPT than the other two algorithms. Moreover, NTA has a better performance than FTA. In the second group of simulations, we change the number of tasks, while keeping the average workload of all tasks fixed. The results of the average makespans are shown in Fig. 6. For each real trace, the average makespans of FTA and NTA are still smaller than those of the two compared algorithms. Moreover, NTA has the best performance besides OPT. In addition, along with the increase of the average workload or the number of tasks, the absolute errors of our algorithms compared to OPT in both groups of simulations have little change. This is because the absolute errors of our algorithms mainly depend on the inter-meeting times between requesters and other users, and these inter-meeting times are actually derived from the real traces so that they keep unchange in the whole simulations. Moreover, when the number of tasks and the average workload increase, the average makespans of all algorithms become larger. Since the absolute errors change little, the ratios of average makespans between our algorithms and OPT are very close to 1. These observations exactly validate our theoretical analysis results.

Second, we evaluate the performances of the five algorithms on the synthetic traces, taking the different number of users, average workload, number of tasks, and average rate parameter into account. When we evaluate the performances for one parameter (i.e., the number of users, the average workload, the number of tasks, or the average rate parameter), we keep the other three parameters fixed. The results are shown in Fig. 7. These results also prove that FTA and NTA have better performances than WF and LF. Moreover, NTA has the best performance among the four algorithms. As in the simulations on real traces, along with the increase of the average workload or the number of tasks, the absolute errors of our algorithms also have little change. Nevertheless, along with the increase of the average rate parameter, the absolute errors of our algorithms become smaller and smaller. This is because the average inter-meeting time decreases, which leads to decreasing absolute errors. In addition, when the number of mobile users increases, the average time for a requester to meet a user also decreases, so that the average makespans of our algorithms decrease. These observations still remain consistent with our theoretical analysis results.

VI. Related Work

In this paper, we focus on the task assignment problem in mobile crowdsensing, which involves a lot of mobile users in solving a large job through their carried mobile devices [9].



(a) Change the number of users (b) Change the average workload (c) Change the number of tasks (d)Change the average rate parameter

Fig. 7. Performance comparisons on the synthetic traces with the different number of users, average workload, number of tasks, or average rate parameter.

By far, there have been several frameworks, platforms, and incentive mechanisms designed for crowdsensing systems [3], [6]–[8], [11], [12], [16]. Among them, the most related work is the LRBA algorithm designed for a task allocation problem in mobile crowdsensing [11]. Unlike our work, the problem in this work is NP-hard, and the tasks are location dependent.

On the other hand, our task assignment problem is also different from traditional parallel machine scheduling problems. In fact, there have been thousands of papers on parallel machine scheduling problems by far. Literatures [2], [5] have made a detailed review on these works. Even in recent years, there is still much research on the complex parallel machine scheduling problems, such as [1], [4]. The most related works among the existing researches are based on the parallel machine scheduling models, taking the setup time into consideration. In these works, each task is assumed to have a common setup time to all machines, but remains diverse for different tasks. Moreover, most of these problems are NP-hard. In contrast, the tasks being sent to mobile users in our problem follow the mobility model. More specifically, each task being sent to a mobile user is a probabilistic event. The probability distribution might be different for mobile users, but remains common for diverse tasks. Moreover, mobile users need to return the result for each task, which is also a probabilistic event. Such a unique task assignment model makes our problem different from existing parallel machine scheduling problems.

In addition, there is also much research in the MSN field, which mainly focuses on routing problems [10], [17]. None of them have studied task assignment problems, except Water Filling, which is proposed to allocate tasks among the mobile users in an MSN [15]. Although this work also takes the delivery time of the tasks and their results into consideration, it mainly focuses on minimizing the latest makespan of all tasks, which is an NP-hard problem, unlike ours.

VII. Conclusion

In this paper, we study the task assignment problem for mobile crowdsensing in MSNs. In order to minimize the average makespan of the assigned tasks, we propose an offline task assignment algorithm FTA and an online task assignment algorithm NTA. Both FTA and NTA adopt a greedy task assignment strategy, but at different time. Theoretical analysis shows that FTA is the optimal offline task assignment algorithm, and NTA can achieve a smaller average makespan in the online decision case. We also give a competitive ratio of NTA, and conduct a series of simulations on real traces and synthetic traces. The results have also proven the significant performance of our algorithms.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (NSFC) (Grant No. U1301256, 61379132, 60803009, 61170058), the NSF of Jiangsu Province in China (Grant No. BK20131174), and NSF grants CNS 149860, CNS 1461932, CNS 1460971, CNS 1439672, CNS 1301774, ECCS 1231461, ECCS 1128209, and CNS 1138963.

References

[1] B. Alidaee and H. Li. Parallel machine selection and job scheduling to minimize sum of machine holding cost, total machine time costs, and total tardiness costs. *IEEE Transactions on Automation Science and Engineering*, 11(1):294–301, 2014.

[2] A. Allahverdi, C. Ng, T. Cheng, and M. Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008.

[3] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola. Fostering participaction in smart cities: A geo-social crowdsensing platform. *IEEE Communications Magazine*, 51(6):112–119, 2013.

[4] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001.

[5] T. Cheng and C. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.

[6] V. Coric and M. Gruteser. Crowdsensing maps of on-street parking spaces. In *IEEE ICDCS*, 2013.

[7] A. Farshad, M. K. Marina, and F. Garcia. Urban wifi characterization via mobile crowdsensing. In *IEEE NOMS*, 2014.

[8] Z. Feng, Y. Zhu, Q. Zhang, L. M. Ni, and A. V. Vasilakos. Trac: Truthful auction for location-aware collaborative sensing in mobile crowdsourcing. In *IEEE INFOCOM*, 2014.

[9] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: Current state and future challenges. *IEEE Communications Magazine*, 49(11):32–39, 2011.

[10] W. Gao, Q. Li, B. Zhao, and G. Cao. Multicasting in delay tolerant networks: A social network perspective. In *ACM MobiHoc*, 2009.

[11] S. He, D.-H. Shin, J. Zhang, and J. Chen. Toward optimal allocation of location dependent tasks in crowdsensing. In *IEEE INFOCOM*, 2014.

[12] X. Hu, T. H. S. Chu, H. C. B. Chan, and V. C. Leung. Vita: A crowdsensing-oriented mobile cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 1(1):148–165, 2013.

[13] J. Burgess et al. CRAWDAD data set umass/ diesel (v. 2008-09-14). Downloaded from http://crawdad. cs.dartmouth. edu/umass/diesel, Sept. 2008.

[14] J. Scott et al. CRAWDAD data set cambridge/ haggle (v. 2009-05-29). Downloaded from http://crawdad. cs.dartmouth. edu/cambridge/haggle, May 2009.

[15] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *ACM MobiHoc*, 2012.

[16] J. Sun. An incentive scheme based on heterogeneous belief values for crowd sensing in mobile social networks. In *IEEE Globecom*, 2013.

[17] J. Wu, M. Xiao, and L. Huang. Homing spread: Community home-based multi-copy routing in mobile social networks. In *IEEE INFOCOM*, 2013.