CrowdBind: Fairness Enhanced Late Binding Task Scheduling in Mobile Crowdsensing

Heng Zhang[1], Michael A Roth[1], Rajesh K. Panta[2], He Wang[1], and Saurabh Bagchi[1]

[1]Purdue University

[2]AT&T Labs Research

Abstract

Mobile crowdsensing (MCS) is an efficient method to collect sensing data from a large number of mobile devices. Traditionally, low task coverage and high energy consumption on mobile devices are two of the main challenges in MCS and they are extensively studied in the literature. In this work, we discuss a third factor, scheduling fairness, which is correlated with the other two factors and has a significant impact on the success of MCS. We propose a new framework, called CROWDBIND, that takes advantage of the late-binding characteristic of crowdsensing tasks in addition to incorporating a trajectory-based mobility prediction model to schedule tasks. We conducted a survey with 96 participants to learn about how users react to varying levels of fairness in MCS applications. We designed and implemented a full-stack MCS system including a scheduling server and an Android client. We evaluate our system by conducting an IRB approved user study of 50 people in our college town for one month as well as running a simulation using Gowalla dataset of 90K users. CROWDBIND is proved to be effective in a large population and the results show that CROWDBIND achieves the highest scheduling fairness compared to prior works (Periodic sensing, PCS, Sense-Aid, and CrowdRecruiter), improves the average per-device energy efficiency from 18.3% to 91.4%, and improves the task coverage from 9.7% to 52.1%.

1 Introduction

Mobile crowdsensing (MCS) [2] is a technique by which sensor data [29] is sourced from a large group of individuals with mobile devices capable of computing and sensing. This data can be used to extract information that is of common interest, such as weather conditions, traffic information, and social network applications [23]. By leveraging the powerful sensing capacity and ubiquity of mobile devices, MCS can provide information about our environment while lowering the cost of running data collection campaigns.

The application and the user (*e.g.,* smartphones and IoT devices) are the two main players in the MCS ecosystem [27]. The application issues sensing tasks to users, who respond with the results after performing the requested actions. An optional scheduler between the application and the user can be added to optimize these interactions. The most common type of task in MCS is the periodic task, in which the same sensing action is requested at regular intervals over some longer duration of time. To facilitate scheduling for these tasks, multiple **Task Instances** are created for each task at its interval of periodicity. Prior works focus on how users can be selected to maximize task coverage under various constraints (*e.g.,* budget, probabilistic coverage) [4, 6] as well as how the client-side energy cost of sensing and uploading data can be reduced [10, 31]. However, we find there is another important factor for the viability of MCS—scheduling fairness. This refers to how equitably the overall task load is divided among all the participating devices. In the short term, unfairness will deplete the device energy of some users who frequently receive tasks and cause them to leave MCS campaign. In the long term it will harm task coverage because of less participants. Therefore, maintaining scheduling fairness is important to keeping users continuously participating in MCS campaigns. However, the study of scheduling fairness in MCS task scheduling area is limited. Some works [11, 13, 14] mention that fairness is a desirable and important property but they do not optimize the fairness.

Importance of Fairness

To understand if the scheduling fairness affects MCS users' willingness to participate in MCS, we surveyed 96 individuals from 11 countries (51% from the United States, 23% from India, and 16% from China). In the survey, We hypothesize 3 different payment models. First, the volunteer model requires users' voluntary participation. Users do not receive a monetary benefit, but they can ask for information shared by other volunteers without paying the information provider. Second, the subscription model pays participants a fixed amount of money for every subscription period no matter how many task instances are assigned to the each user. Third, the pay-by-work model pays participants based on their relative contribution in terms of the tasks completed

International Conference on Embedded Wireless Systems and Networks (EWSN) 2020
17–19 February, Lyon, France
ISBN: 978-0-9949886-4-5

or volume of data uploaded. These three payment models represent our assimilation and supersetting of models proposed in various previous works [12, 21].

For each of the three models, we asked the users the following questions: *1) Do you think it unfair if you receive more tasks under this payment model* and *2) will you drop MCS because of the unfairness under this payment model*. The results in Figure 1 demonstrate that an unfair task allocation could cause 64.9% of the users to drop out of MCS campaign in the subscription model and almost half (48.1%) of the users to drop out of the volunteer model. We were especially surprised by the results for the pay-by-work model, in which we hypothesized that unfairness would not be a problem because a user would receive payment in proportional to their costs. However, more than $1/3$ of the users were likely to leave the MCS campaign because of the unfair task distribution.

Three crucial MCS criteria (reward, detour, and energy consumption) have been extensively studied in the literature. Reward refers to how much the participant can benefit from MCS (e.g. money or interested information). Detour refers to that people may be asked to move to a certain area that detours from the user's original route in order to sense the value at the detoured area of interest. Energy consumption essentially means how much energy is consumed by the MCS smartphone applications. We also want to know whether fairness is a fundamentally important criteria. Therefore, we further ask the users to rate the importance of each of the four criteria with 4 options: *Very Important*, *Important*, *Not Important*, and *Do Not Know*. The results are summarized in Figure 2. 47% of the users rate fairness as very important, which is the highest among all criteria. 80% of users rate fairness as important or very important. As a comparison, this number is 83% for the reward, 80% for detour, and 74% for energy consumption. Understandably, MCS users wish to be fairly assigned with tasks hence fairness is perceived to be a crucial criteria.

One may argue that even if fairness is perceived to be important by end users, how will participants in an MCS campaign know whether it is fair or not. For one, we believe philosophically that this property is desirable independent of its observability. Second, unfair allocation can cause users' mobile devices to run out of energy leading them to leave the MCS campaign. Third, many campaigns have users that belong to a cohort or a social group and therefore it is likely that out-of-band communication can lead users to determine the fairness (or otherwise) of the task allocation.

Our Design Features

We find a way to optimize the scheduling fairness without degrading two other primary properties—task coverage and total energy requirement to accomplish a task. In our design, we model fairness using Gini coefficient [26] and introduce design features that improve the estimated metric. Five opportunities are utilized to improve the task coverage and energy efficiency of mobile devices in a fair manner of MCS task scheduling. **First**, MCS tasks have a late binding property. At the instant a task is submitted to our framework, our solution does not need to map all the instances of this task to the devices right away. Rather CROWDBIND can

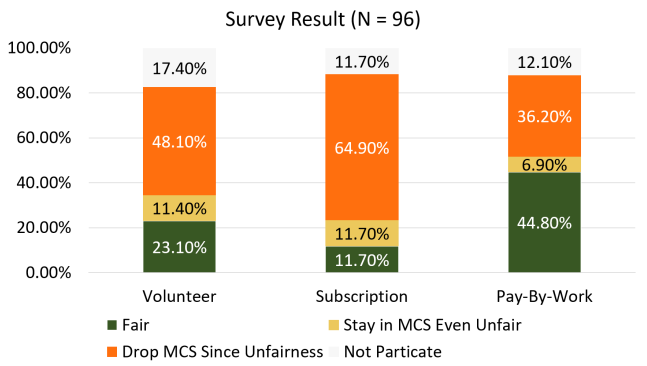


Figure 1: Result of survey on attitudes towards fairness in 3 different payment models. A total of 96 users from 11 countries participated in the survey.

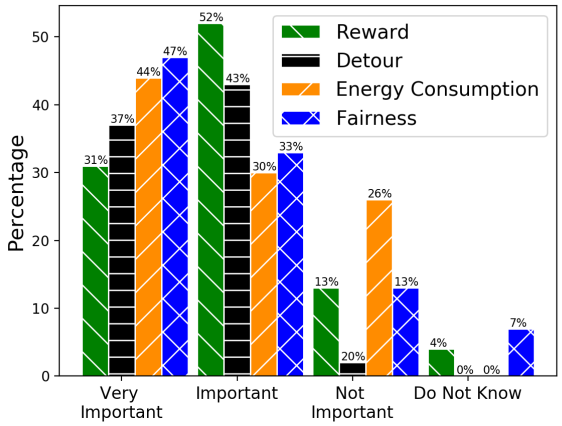


Figure 2: Result of survey on the relative importance of 4 MCS criteria.

wait till shortly before the starting time of the task instance to determine which clients can satisfy it. **Second**, MCS tasks have some delay tolerance, *e.g.,* the barometer reading at our campus quad can be taken now or within the next few minutes when more people come into the region without sacrificing the accuracy of the aggregate results. **Third**, MCS tasks have some spatial tolerance, *i.e.,* a device anywhere within some radius of the task region can perform the task. **Fourth**, regularity exists in human mobility patterns [18]. According to Thuillier [19], 82.75% of individuals can be clustered into 12 categories of mobility patterns. The mobility pattern can be utilized to predict potential participants for a task. **Last**, building a global view of different task instances in a certain lookahead window and utilizing the late binding and the delay tolerance properties help improve the three optimization goals (scheduling fairness, task coverage, and energy efficiency). Because the scheduler knows the requirements of task instances and the availabilities of more users (task coverage) ahead of time, it can reduce the repeated use of some users who are eligible for multiple task instances (energy efficiency and fairness).

We propose CROWDBIND that looks ahead and finds out potential users for a task within its delay tolerance and task region. Two variants of greedy algorithms are designed.

Greedy-Heuristic (G-Heuristic for short) in Section 3.2.3 achieves fast convergence with relatively high fairness and task coverage. It can be used in scenarios where fast scheduling is needed. Greedy-Random-X (G-Random-X for short) in Section 3.2.2 achieves better fairness at the cost of longer convergence time. This will be the algorithm used in general MCS campaigns. CROWDBIND is able to estimate the likelihood of a user entering the task region based on the user mobility prediction and builds a fair schedule to determine *which* devices should be selected to perform *which* task instances without some users to be frequently tasked.

We conducted a one-month user study (approved by IRB) with 50 students on our college campus to evaluate our framework. We also run simulation using the real-life Gowalla dataset [17] with 90K users to evaluate the effectiveness of our framework in a large population. We compare with 3 competing baselines: PCS [10], CrowdRecruiter [28], Sense-Aid [31], and one naïve baseline: Periodic [3]. We evaluate CROWDBIND under varying task radii, delay tolerance, and MDR (defined in Section 2) of the task. In the user study, by polling at high frequency, Periodic can achieve the highest task coverage while CROWDBIND achieves almost the same coverage as Periodic but, on average, uses only 29.3% of the per-device energy. CROWDBIND also has the best scheduling fairness over all frameworks with an improvement from 11.6% to 71.2% and the best average energy efficiency with an improvement from 18.3% to 91.4% over the four solutions. The simulation results are consistent with the user study results which proves the effectiveness of our framework in a large population use case. Note that in the rest of the paper, we use the term *user*, *client*, and *device* synonymously.

Our contributions in this work are summarized as follows:

1. We propose two greedy algorithms to solve the fairness optimization problem. To the best of our knowledge, this is the first work to optimize scheduling fairness in the area of MCS. The two greedy algorithms have their own distinct advantages to be used in a complementary manner in real life deployment to satisfy different application requirements.
2. Our design leverages 5 insights about the nature of MCS to improve all 3 MCS metrics: fairness, task coverage, and average energy efficiency. Our framework follows the event-driven design principle as opposed to the polling-based approach in order to improve the scalability of the scheduler.
3. We conducted a real-world MCS campaign on our university campus over one month with 50 users to evaluate CROWDBIND and the 4 competing solutions. The collected mobility dataset is available at [30].

The rest of the paper is structured as follows. Section 2 provides the insights that lead to our design. The design details of CROWDBIND is described in Section 3. We compare it with the other four solutions in Section 4. Section 5 talks about the related works. The paper discusses the limitations in Section 6 and concludes with Section 7.

2 Motivation

To see how the three optimization goals are achieved by utilizing different properties in MCS, we start with the following definition. We use **Minimum Device Requirement (MDR)** to denote the fewest number of users that are required by a task instance and use **Task Coverage** to determine if a task is satisfied. A task is *satisfied* if its coverage is greater than a minimal value specified by the task. We use minimum coverage requirement (MCR) to denote this minimal value (*e.g.,* MCR = 0.8 means the task needs a minimal task coverage of 80%). Equation 1 is the formula for task coverage:

$$\text{Task Coverage} = \frac{1}{N}\sum_{i=1}^{N}\frac{\min(S_i, MDR)}{MDR} \tag{1}$$

where i is the task instance index, N is the total number of task instances for a given task, and S_i is the number of selected devices for task instance i. Task coverage is calculated as the average coverage of each task instance ranging from 0 to 1, *i.e.,* a two-instance task with MDR of 4 and MCR of 0.8 would be satisfied when its task coverage equals $(4/4+4/4)/2 = 1 > 0.8$ if each instance finds 4 users, while the task coverage is reduced to $(2/4+4/4)/2 = 0.75$ if one of the two instances finds only 2 users so the task is not satisfied since the task coverage is lower than its MCR.

An Illustrative Example

Mobility prediction and lookahead window can help improve the task coverage, the scheduling fairness, and the average energy efficiency. In the example of Figure 3, A task has two instances, TI_1 and TI_2, each of which has MDR = 2. By looking ahead and predicting users' mobilities, CROWDBIND can predict U_1, U_2, and U_3 stay in the whole delay tolerance window of TI_1 (from t_1 to t_2), U_2 and U_3 leave the task region at t_2, U_1 leaves the task region at t_4, and U_5 walks into the task region at t_4. Therefore, CROWDBIND will assign TI_1 to U_2 and U_3 and TI_2 to U_1 and U_4. Each user is only selected once. If not using mobility prediction but periodically polling users' location, there is no guarantee that a poll of U_4 will happen between $t4$ and $t5$ thus TI_2 will only have U_1. Therefore the task coverage is hindered. If not looking ahead, when the scheduler is selecting users for TI_1, because it does not know the future information of TI_2, it will randomly choose two users for TI_1 and U_1 has 2/3 chance to be selected. Since U_1 will definitely be selected for TI_2, eventually, the final selection decision has 2/3 chance to select U_1 twice, which leads to an overall Gini coefficient of 0.375. This is not as fair as the selection by CROWDBIND which gives an overall Gini coefficient of 0 (fairest). Additionally, given the same amount of total tasks, a fair scheduling decision will evenly distribute the whole workload to a larger population. Therefore, the average energy cost among the selected users will be lower in a fair scheduler because more people are selected to serve the same amount of workload. For example, U_1 has 2/3 chance to work more than others thus the energy cost of U_1 is higher. This will not happen in the selection by CROWDBIND.

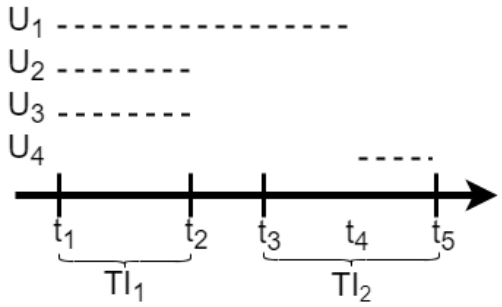


Figure 3: Example to show how the MCS properties benefit task coverage, fairness, and average energy efficiency.

3 Design

In this section, we describe the structure of CROWDBIND server, which is the scheduling orchestrator between the MCS applications and the clients. Figure 4 shows the architecture of CROWDBIND server as well as its relationship with the MCS applications and the clients. CROWDBIND schedules some tasks, which are submitted by MCS applications, for the clients to perform. The specifications of these tasks are shown in Table 1. Note that all the parameters except the start time and end time are the same for all instances of a given task. Once the device receives a task instance from CROWDBIND, it immediately samples the requested sensors and replies with the sensed data. Once the tasks have been successfully completed, CROWDBIND will send crowdsensed results to the MCS application. In our system, most of the workload is relegated to the server, while the mobile client mainly responds to scheduled task instances.

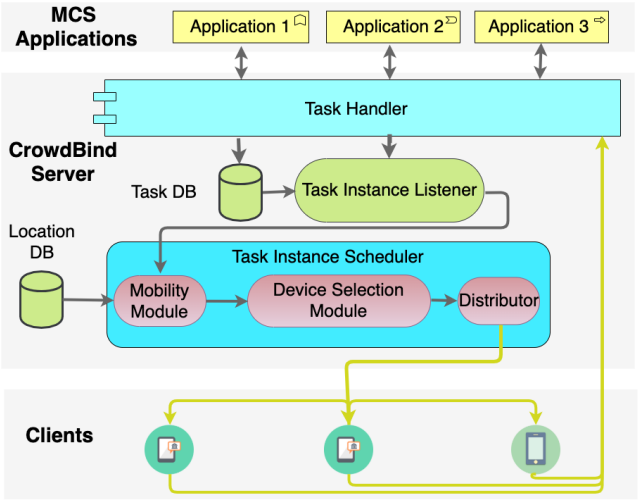


Figure 4: CROWDBIND Architecture. It shows the MCS application, the mobile client app, and the CROWDBIND server.

3.1 CROWDBIND Server

The CROWDBIND server handles incoming tasks from the MCS applications. It has several components.

3.1.1 *Task Handler*

The task handler is responsible for parsing the tasks received from the MCS applications and replying with the results. According to the periodicity and time requirement for each task, the handler will create multiple instances of it and store them into the task database (Task DB). For example, the task handler will generate 6 instances of a task starting at 8 a.m. and ending at 9 a.m. with a periodicity of 10 minutes. After the insertion of new task instances, the task handler will notify the task instance listener.

Parameters	Description
Location	The center of the task, specified in geographic coordinates
Radius	The radius of a circle inside which sensing data is requested
Start Time	The starting time of the task
End Time	The ending time of the task
Periodicity	The time interval between two successive task instances
Delay Tolerance	The task instance's results are useful if received between the start time of that instance and the start time + the delay tolerance value
Sensor List	The sensors required by the task
Minimum Device Requirement (MDR)	The number of devices required by each instance of the task
Minimum Battery Requirement (MBR)	The minimum battery level for a device to be qualified for sensing and uploading data
Minimum Coverage Requirement (MCR)	The minimum coverage required, averaged over all instances for the task

Table 1: Parameters of a crowdsensing task

3.1.2 *Task Instance Listener*

The task instance listener monitors new task instances and deliver them to the task instance scheduler. The listener maintains a L-minute lookahead window and checks task instances in the Task DB whenever it is notified by the task handler. If it finds any instances that are within the lookahead window, it will deliver them to the scheduler and sleep until it is notified again or until the time of the next task instance that is outside of the current lookahead window. For example, in Figure 5, the task listener is notified at time t_s and it looks for task instances whose times lie in $[t_s, t_s + L]$. There are three such instances: TI_1, TI_2, and TI_3, which have delay tolerances Δd_1, Δd_2, and Δd_3 respectively. It then delivers TI_1, TI_2, and TI_3 to the scheduler and goes to sleep until the time of the next instance or if the task handler notifies it due to the arrival of new task instances from some MCS applications. A trade-off exists in determining the value of L. A larger L includes more task instances and provides CROWDBIND with more flexibility to schedule them among available devices. However, the accuracy of the mobility prediction suffers with a large L. In our evaluation, we choose L as 20 minutes to best balance these two factors given the performance of mobility prediction algorithm.

3.1.3 *Task Instance Scheduler*

The task instance scheduler is in charge of selecting devices for each task instance and allocating sensing work to those devices. It consists of three modules: *the mobility module, the device selection module*, and *the distributor*. The scheduler is idle until it receives task instances from the task instance listener with the instances being sorted by their starting times. In the example of Figure 5, the instances are

ordered: TI_1, TI_2, TI_3. The scheduler will loop through all the instances and find the corresponding available devices by mobility prediction. For example, for TI_2, it will populate the list of the users who are expected to appear in the task region between $(t_2, t_2 + \Delta d_2)$.

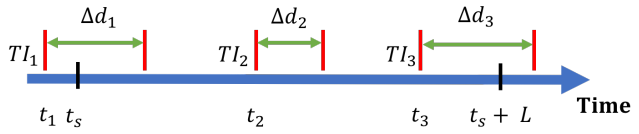


Figure 5: A lookahead window of length L contains three task instances. The scheduler will use the lookahead and map each instance to the devices that can potentially satisfy that instance.

The mobility module and the distributor

We adapt a trajectory-based mobility prediction algorithm from [9] that extrapolates the future locations of a user from the most similar historical trajectory. The mobility prediction model achieves average 75% accuracy in predicting the future locations 20 minutes in advance of our users (see Section 4 Figure 6).

The distributor will send the scheduling decision to devices at the time of the task instance. It will wake up as mandated by the scheduler and inform the selected devices, *e.g.,* for TI_2, wakeup will be in $(t_2, t_2 + \Delta d_2)$. When sending the task instance, the scheduler will double check the current location of the selected devices. If a device is not present in the task region—which will happen if the mobility prediction is inaccurate—the mobility predictability (P_u in Table 2) of the device will be decreased and the scheduler will execute the G-Heuristic algorithm (more details in Section 3.2) to quickly find the replacements. Because of the late binding property, the delay tolerance property, and the runtime efficiency of the G-Heuristic algorithm, the scheduler usually has enough time to find replacement devices.

Task Coverage and Fairness through Mobility Prediction and Lookahead Window

We now analyze how the mobility prediction helps improve task coverage as well as how the mobility prediction together with the lookahead window help improve scheduling fairness. We define Eligibility Ratio (ER) in Equation 2 to aid the following analysis.

$$\text{ER} = \frac{\text{\# Available devices}}{\text{MDR}} \tag{2}$$

Analysis for task coverage

If $ER < 1$, fewer than MDR number of clients can be found so the task instance is *unsatisfied*. On the other hand, when $ER >= 1$, the larger the ratio is, the easier it is for the scheduler to satisfy the task instance and the scheduling decisions will be fairer. Mobility prediction fundamentally increases ER because CrowdBind can predict those users who are currently not in the task region but will appear in the region within the delay tolerance of the task. The accuracy of mobility prediction affects the ER because the true number of available devices is not known to the scheduler. Nevertheless, a good predictor will give a number close to the ground truth and thus increase ER.

Analysis for fairness

The benefits of lookahead come from the ability of knowing future tasks and their requirements as well as the knowledge—brought by mobility prediction—of who will be eligible for which tasks. The lookahead itself cannot help the task scheduling because even if the scheduler knows that tasks will be scheduled in the future, it does not know who can satisfy those tasks therefore it cannot fairly and efficiently distribute tasks. As shown in the example of Figure 3, suppose the mobility prediction correctly predicts the locations of those 4 devices, then CROWDBIND can guarantee that the fairest mapping between the instances and the devices is achieved (as long as all devices have equal conditions for usage, battery, and sensor integrity). CROWDBIND knows which are the eligible devices that can satisfy the task instances in the lookahead window and it has the information about how many times devices have been selected in the past. Thus, it can fairly assign task instances to devices.

In comparison, the approach without lookahead cannot guarantee to achieve the fairest mapping because it does not know the future and therefore cannot "keep in reserve" U_2 and U_3 for task TI_1 in the example of Figure 3.

Consider a quantitative reasoning of the probability of achieving perfect fairness. Suppose there are N task instances arriving in the order of instance 1 to N. Each of the instances requests m clients and the available clients for each task instance are $Nm, (N-1)m, (N-2)m, ..., m$ respectively. Thus, task instance 1 is the easiest to satisfy since it has Nm available users while instance N has to choose exactly the m available users. The probability of achieving the fairest mapping for these task instances by a baseline scheduler, *i.e.,* one without any lookahead, is given by Equation 3.

$$P_{fairest} = \frac{1}{\prod_{i=1}^{N} C((N-i+1)m, m)} \tag{3}$$

The insight is that there is only one specific choice to achieve the perfect fairness. The number of possible ways for task instance 1 to select the m users is $C(Nm, m)$, for instance 2 it is $C((N-1)m, m)$ and so on till instance N whose $C(m, m) = 1$. CROWDBIND can *deterministically* achieve the perfect fairness, provided that there is no inaccuracy in the prediction and the task instances are all within the lookahead window.

3.2 Device Selection Module

Searching for a set of MDR users out of all possible combinations of K available users under certain constraints (*i.e.,* budget and probabilistic coverage) in MCS has been proven to be NP-hard [7, 15]. The brute-force approach is to try all possible combinations of user selections and choosing the fairest selection. However, this approach is not computationally efficient. Although one task instance may only need a few devices, the total number of task instances and the total number of available devices for each task instance could be huge within a lookahead window, which will lead to a large number of different combinations. In this work, we propose two variations of greedy selection algorithms that consider more comprehensive requirements as summarized in Table1 to select users for task instances. We call these two algorithms **G-Heuristic** and **G-Random-X**. Before discussing the details of G-Heuristic and G-Random-X, we first formulate the selection problem.

3.2.1 Problem Formulation

We formulate the problem as a minimization problem with three constraints. First, the task needs to specify the *MBR*, which is the minimum battery level that a device should have to be eligible for that task. The purpose of this constraint is to reduce the energy impact of MCS tasks on the device battery. This is particularly pertinent for the MCS tasks that are energy hogs such as taking and uploading pictures. Second, the task needs to specify the *MDR*, which is the minimum number of devices that are required for the completion of each instance of this task. The last constraint is the **Minimal Completion Coverage (MCC)**, which essentially means the minimum number of devices that will be involved in the completion of the whole task. $MCC = MDR \times MCR \times N$ where *MCR* is the minimal task coverage that the task requires and N is the number of instances of a task. For example, a task with $MCR = 0.8$ expects that the average task coverage as calculated by Equation 1 will be greater than or equal to 0.8 after all of its instances are done. If the task has $MDR = 4$ and 5 instances, its *MCC* equals $4 \times 0.8 \times 5 = 16$. This means the task will have to select at least 16 devices to satisfy its minimal coverage requirement. By calculating the probability of a task instance to be completed by a user ($PC_u(ti)$ in Equation 5), the **completion coverage (CC_t)** of a task in Equation 6 is the expected number of users that will be involved in sensing activities related to the task, among all of its instances.

The problem is then formulated as follows (notations are in Table 2):

$$\begin{aligned} &Minimize \;\; Gini(\mathcal{U}) \\ &s.t. \;\; \forall_{u \in U_{ti}, ti \in \mathcal{T}I} \quad u.battery \geq MBR_{ti} \\ &\qquad \forall_{ti \in \mathcal{T}I} \qquad\qquad \|U_{ti}\| == MDR_{ti} \\ &\qquad \forall_{t \in \mathcal{T}} \qquad\qquad\quad CC_t \geq MCC_t \end{aligned} \tag{4}$$

Within the lookahead window, CROWDBIND has a global view of the set of task instances $\mathcal{T}I$ and the set of users $\mathcal{U}$ that are predicted to be available for at least one of the task instances. So it seeks to maximize the fairness of the choice of users, *i.e.,* minimize the Gini index. The fairness should have lower priority than task coverage and device energy efficiency in the optimization. We only optimize the fairness if we predict that the *MCC* will be satisfied and that the selected users will not experience an unacceptable battery drain. For each task instance $ti \in \mathcal{T}I$, its set of available users is denoted as $\mathcal{U}_{ti}$. From $\mathcal{U}_{ti}$ we attempt to find a subset, $U_{ti} \subseteq \mathcal{U}_{ti}$ for task instance ti, such that $\|U_{ti}\| == MDR_{ti} \leq \|\mathcal{U}_{ti}\|$. (No need for optimization if $\|\mathcal{U}_{ti}\| \leq MDR_{ti}$). The first two constraints are directly related to each individual task instance whereas the last constraint is related to all instances of a task. Note that the total energy consumption for a crowdsensing task is not increased due to CROWDBIND because for a generic task with N instances and requiring K users for each instance, there will be a maximum of $N \times K$ data uploads with or without our protocol. For each data upload, the client can use any energy saving method (*e.g.,* piggybacking [10] or uploading within cellular tail time [31]) to reduce the energy cost for the individual upload.

$$PC_u(ti) = P_u \cdot f_u(ti) \tag{5}$$

$$CC_t = \sum_{ti \in t.instances} CC_{ti} = \sum_{ti \in t.instances} \sum_{u \in U_{ti}} PC_u(ti) \tag{6}$$

Our two variants of greedy algorithms have two stages: the initialization stage and the optimization stage. As required by the third constraint, we need to know the expected number of users in a task (CC_t). Therefore, we also need to know the expected number of users involved in each of the instances of a task. But because task instances occur sequentially and because fairness is based on the utilization of users—which can only be known after we have some selection decisions—we use an initialization stage to initialize the selection decisions from which the scheduler can further optimize the fairness. The two variants only differ in their initialization stage.

Symbol	Meaning
$\mathcal{T}$	The set of tasks in the lookahead window
$\mathcal{T}I$	The set of different instances of tasks from $\mathcal{T}$
$\mathcal{U}$	The set of available users who are available to at least one task instance from $\mathcal{T}I$
U	The set of users who are selected by at least one task instance from $\mathcal{T}I$ ($U \subseteq \mathcal{U}$)
t	A task from $\mathcal{T}$
$t.instances$	The set of instances generated by task t
$\mathcal{U}_{ti}$	The set of available users for task instance ti
U_{ti}	The set of selected users for task instance ti. ($U_{ti} \subseteq \mathcal{U}_{ti}$)
u	A user
$u.battery$	The battery level of device u
$u.s$	The utilization value of user u. s equals to the number of times a device has been selected for task instances
P_u	The mobility predictability of user u
$f_u(ti)$	The sensor integrity score of user u to satisfy the sensor requirement of task instance ti (from 0 to 1)
$PC_u(ti)$	The probability of user u to complete task instance ti
N_u^{ti}	The number of task instances that user u can satisfy within the lookahead window
$W_u(ti)$	The effective weight of user u to complete task instance ti (from 0 to 1)
CC_t	The completion coverage of task t (from 0 to $MDR_t \times$ the number of instances of this task)
MCC_t	The minimal completion coverage of task t ($MDR_t \times MCR_t \times$ the number of instances of this task)

Table 2: List of notations used in design

3.2.2 Greedy-Random-X

The Greedy-Random-X algorithm (G-Random-X for short) uses a random initialization method. For each task instance ti in the lookahead window, G-Random-X randomly initializes a set of selected users U_{ti} from the available users in $\mathcal{U}_{ti}$ that satisfy the MBR. X denotes how many random seeds are used to initialize the selection decision in the initialization stage. This initial selection may not satisfy some of the constraints and the Gini coefficient may not be the lowest achievable value in the current lookahead window because the selection may be locally optimal. Therefore, the optimization stage greedily replaces some users in selection decisions to satisfy any unsatisfied constraints as well as to minimize the Gini coefficient. The fact that the selection is only locally optimal is a consequence of only selecting users for each task instance individually. A change in the location of the users within an instance may cause the same users to be used later in other instances. Therefore, we run the algorithm X times using X different seeds to choose the best

selection that gives the lowest Gini Coefficient out of all X selections.

3.2.3 Greedy-Heuristic

The Greedy-Heuristic algorithm (or G-Heuristic for short) has a more complex initialization stage. The reason behind the complexity of this algorithm is that we want it to reach the optimal selection faster by a smart selection of initial users.

For the sake of fairness, we record the number of times that each user has been selected for sensing ($u.s$ in Table 2). If a user has been selected a high number of times, the user will be less preferable. To satisfy a task instance, the user needs to have a high predictability as well as having the required sensors that can generate reliable data (P_u and $f_u(ti)$). We calculate $f_u(ti)$ using the sensed data history and the value $\in [0,1]$, where 1 denotes perfect reliability. Based on the predictability of mobility P_u and the sensor integrity score $f_u(ti)$ of a user for a specific task instance, we calculate the probability of a task instance to be completed by this user $PC_u(ti)$ using Equation 5. The higher the value, the more likely the user can satisfy the task instance. When choosing users for each task instance, G-Heuristic prefers users who can satisfy the least number of task instances (N_u^{ti}) to increase the chance of a later task instance being satisfied by a user who has not been tasked already. For example, if a task instance TI_1 can be satisfied by users u_1 and u_2 and TI_2 only by u_2, then G-Heuristic will reserve u_2 for TI_2 and u_1 for TI_1. The final decision on which user to choose is based on the **effective device weight** $W_u(ti)$ which denotes the weight of a device bidding for a task instance (Equation 7). The higher the weight is, the more likely that the scheduler will choose that device. It means the algorithm prefers a user with high probability to complete a task instance and low number of task instances that it can satisfy.

$$W_u(ti) = \frac{1}{N_u^{ti}} \cdot PC_u(ti) \tag{7}$$

Eventually, G-Heuristic initializes the set of selected devices for a task instance ti by choosing MDR number of users in descending order of the **selection weight**, SW of each user which is defined in Equation 9. u' denotes the same user as u but the utilization value $u.s$ is incremented by 1. $\mathcal{DIFF}$ in Equation 8 is interpreted as the fairness improvement brought by user u. We want to maximize the fairness improvement by selecting the user with positive fairness improvement. It is also possible that a user has a negative fairness improvement (bottom of Equation 9). If no candidates can bring positive improvement, we prefer a user u who minimizes this negative fairness impact. We also want a user with high effective weight therefore the negative fairness impact is divided by the user's effective weight.

$$\mathcal{DIFF} = Gini(\mathcal{U}) - Gini(\mathcal{U} \setminus u \cup \{u'\}) \tag{8}$$

$$SW(u) = \begin{cases} \mathcal{DIFF} \times W_u(ti) & \text{if } \mathcal{DIFF} > 0 \\ |\frac{\mathcal{DIFF}}{W_u(ti)}| (u \text{ worsens fairness}) & \text{if } \mathcal{DIFF} < 0 \end{cases} \tag{9}$$

3.2.4 Optimization Stage

The optimization stage is the same for both G-Random-X and G-Heuristic algorithms. After the initialization, for those instances with a number of available devices larger than MDR (if $||\mathcal{U}_{ti}|| = ||U_{ti}|| \leq MDR$, there is no degree of freedom to optimize the fairness), the optimization stage will be executed. For each task instance, if the parent task of this task instance has $CC_t < MCC_t$, the scheduler will replace the user u whose $PC_u(ti)$ is the lowest in U_{ti} with a user v from $\mathcal{U}_{ti} \setminus U_{ti}$ whose $PC_v(ti)$ is highest. $PC_v(ti)$ must be greater than $PC_u(ti)$ otherwise there is no point in swapping users u and v and the optimization will end.

After the $CC_t \geq MCC_t$ is satisfied and $||\mathcal{U}_{ti} \setminus U_{ti}|| > 0$, we are ready to optimize the fairness. The scheduler optimizes the fairness by choosing a user u_1 whose $u_1.s$ is the lowest in $\mathcal{U}_{ti} \setminus U_{ti}$ to replace a user $v \in U_{ti}$ whose $v.s$ is the highest meanwhile this replacement cannot cause $CC_t < MCC_t$. It first iterates over all $u_i \in \mathcal{U}_{ti} \setminus U_{ti}$ to see if v can be replaced. Then it iterates over all $v \in U_{ti}$.

Algorithms 1 and 2 present the pseudo code of the two greedy algorithms. The iterative process dominates the computation. The worst case happens when it has to go through all available users in $\mathcal{U}_{ti}$ to replace all selected users in U_{ti} for all task instances. The worst case running time is $O(P \cdot Q \cdot MDR)$ where P is the average number of devices in $\mathcal{U}_{ti}$ for all instances, Q is the number of task instances and MDR is the average number of minimum device requirement for those instances.

Algorithm 1: G-Random-X Greedy Algorithm

Input : $\mathcal{TI}$: a set of task instances in one lookahead window, $\mathcal{U}$: a set of users predicted to appear in some regions of the task instances
Output : A mapping between each ti with its selected devices U_{ti}
Randomly initialize the user selection set U_{ti} for every instance ti in $\mathcal{TI}$.
The optimization stage happens only when the number of available users of a task instance ti is larger than MDR.
If the MCC of task is not satisfied, find users with highest completion coverage CC to replace the user with the lowest CC in set U_{ti}.
If the MCC is satisfied, find a user u to replace another user p in U_{ti} such that the selection of u will not cause the MCC to be unsatisfied, the utilization value of u is the highest among those available devices who have not been considered, and user p has the lowest utilization value among those who have been selected.
Repeat the initialization and optimization for X rounds and choose the fairest result.
return U_{ti} for each ti in $\mathcal{TI}$

Algorithm 2: G-Heuristic Greedy Algorithm

Input : $\mathcal{TI}$: a set of task instances in one lookahead window, $\mathcal{U}$: a set of users predicted to appear in some regions of the task instances
Output : A mapping between each ti with its selected devices U_{ti}
Initialize the user selection set U_{ti} for every instance ti in $\mathcal{TI}$.
Initialization selects the users in the descending order of selection weight, SW of each user. The SW of a user is calculated in Equation 9.
The optimization stage is the same as that in Algorithm 1.
Repeat until no replacement can be further executed.
return U_{ti} for each ti in $\mathcal{TI}$

4 Evaluation

We deployed CROWDBIND server on a machine with an Intel Xeon E5-2440 CPU with 12 cores each of which has a cache size of 15360 KB and a clock speed of 1399 MHz. The memory size is 48 GB. There are three parts in the evaluation. In part A, we discuss the trade-offs among the greedy algorithms. Part B shows the user study results comparing with the other 4 baseline solutions. In part C, we run simulation using the real-life Gowalla dataset [17].

The user study (from February 7th to March 7th, 2018) is conducted over a square area of $10km^2$ in a college town with 50 students using Android phones with the various MCS frameworks running on them. All users install five crowdsensing applications: the Periodic protocol, PCS, Sense-Aid, CrowdRecruiter, and CROWDBIND. Users in this study are undergraduate and graduate students from different majors in our university. Users move as usual over the one month period. Location data is received periodically from each device every 5 minutes. The energy cost of location data collection can be optimized by accessing the location information from production cellular provider but understandably we are not allowed to do so. For each crowdsensing protocol, we assigned a total of 213 tasks and 3,916 task instances. Table 3 summarizes the parameters of those tasks. We set the MBR and MCR to be 50% and 0.8 respectively for all the tasks. In all of the task instances, we requested pressure, light, magnetic field, and gravity data from the devices. The size of a single data upload is about 600 bytes.

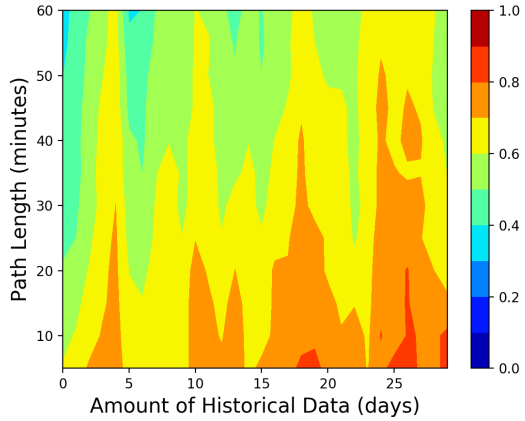


Figure 6: Accuracy of the mobility prediction model for 50 users. This prediction is achieved without the need to build a model through training. Accuracy of above 60% is achieved with only 3 days of historical data.

Figure 6 shows the mobility prediction results for all users in the user study. When using 1, 2, and 3 weeks of history to predict the future 5 minutes to 20 minutes location of the user, the average accuracies over all 50 users are 64.5%, 72.4%, and 74.4% respectively (averaged over the data points for 5, 10, 15, and 20 minutes lookahead).

The four competing solutions that we compare with are briefly described here. **Periodic** sends all the tasks to each device. At a fixed periodicity, each device returns the results for any task instance that the device can satisfy [3]. This approach achieves the highest task coverage but suffers from low energy efficiency because the device may wake up the cellular communication module only to send a small amount of sensor data. Second, **Piggyback CrowdSensing (PCS)** [10] minimizes energy consumption of a device by piggybacking MCS packet on a regular network packet. PCS also sends tasks to all devices and each device will decide if it needs to do the sensing work. Third, **CrowdRecruiter** [28] selects the same users for all instances of a task because it follows a design principle to select the least amount of users who can satisfy the most amount of task instances. The rationale behind this principle is that each participant is paid the same amount of money in one subscription period and choosing less people helps reduce the total incentive payment. Finally, **Sense-Aid** [31], selects only a subset of the available devices in a task region by repeated querying user locations and does not have an estimation of devices availability in the future, which leads to an inefficient use of server resources.

Part A: Compare Heuristic Algorithms

In this section, we compare the two proposed algorithms. G-Random-X is configured with different values of "X". The result is shown in Figure 7. There are 9 tasks with different task regions. For all of them, task radius = 500 meters, delay tolerance = 10 minutes, periodicity = 10 minutes, 6 instances per task, MDR = 3, and MCR = 0.8. Due to the difference in the task locations, the total number of available users are different. The same 9 tasks are repeated 100 times to draw statistically valid results. When the search space is small, the value of X would be low because the convergence will occur easily. The running times of G-Random-X algorithms are proportional to X and the running time of G-Heuristic is similar to G-Random-1. G-Random-10 and G-Random-20 achieve the best task coverage with less outliers and the distributions of the their samples are more stable and convergent to their medians. The G-Random-1 has the worst stability because of the randomness compared to other Xs. The fairness among all algorithms are comparable since they all try to swap users to guarantee fairness. G-Random-10 would be the most practical choice since its high task coverage and fast convergence. The G-Heuristic algorithm is significantly faster with relatively stable and high task coverage. Thus, in a scenario where fast scheduling and task coverage stability are required (*e.g.,* finding the replacement for mis-predicted user and reassign the task), G-Heuristic is a good choice.

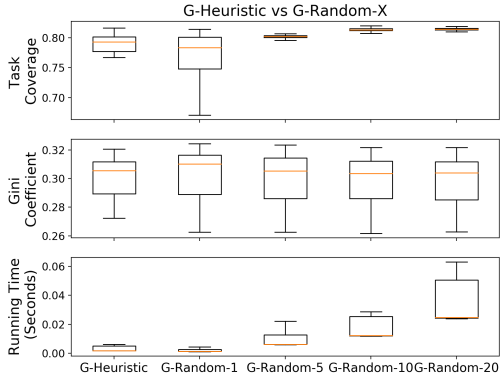


Figure 7: Performance compared among the G-Heuristic algorithm and G-Random-X with various values of "X".

Part B: Overall User Study Results

In this section, we compare our framework with prior works in the user study. G-Random-10 is used in scheduling because it converges fast in the scope of our user study (50 users) and achieves the higher task coverage and better fairness than G-Heuristic.

Varying Parameters	Default Parameters (**MBR** = 50%, **MCR** = 80%)
Task Radius (100, 250, 500 *m*)	**# Locations**: 15 **Start - End Time**: 9 a.m. - 5 p.m. **Periodicity**: 10 minutes **Delay Tolerance**:15 minutes **MDR**: 3
Delay Tolerance (10, 30, 60 minutes)	**# Locations**: 15 **Start - End Time**: 9 a.m. - 5 p.m. **Radius**: 250 *m* **Periodicity**: 10 minutes **MDR**: 3
MDR (1, 2, 5, 8, 10, 15)	**# Locations**: 15 **Start - End Time**: 9 a.m. - 5 p.m. **Radius**: 1 *km* **Periodicity**: 10 minutes **Delay Tolerance**: 15 minutes
# of Concurrent Task Instances (50, 100, 500)	**# Locations**: 5 **Start - End Time**: 9 a.m. - 9:10 a.m. **Radius**: 1 *km* **Periodicity**: the task does not recur **Delay Tolerance**: 10 minutes

Table 3: Settings of Tasks in Part B

Task Coverage

We show the task coverage achieved by various protocols in Figure 8. The Periodic protocol has perfect task coverage because it polls the locations of the devices at a high frequency (at the expense of energy efficiency) and every available device is expected to send the sensor data; thus, it mitigates data loss or devices being switched off. Despite the significant advantage of CROWDBIND over Periodic in client energy efficiency, it remains close behind Periodic in task coverage. The small decrease of CROWDBIND can be attributed to a mis-prediction of a user's location. For example, if we predict a user will be within the task radius but she is not, this will negatively affect our task coverage. Sense-Aid, PCS, and CrowdRecruiter have similar task coverages because they are all implemented by polling the users at the same fixed frequency and all three of these frameworks underperform compared to CROWDBIND because they are unable to capture high-mobility users.

Consider the Eligibility Ratio (ER) and its bearing on the ability of CROWDBIND to achieve high task coverage and fairness. Figures 8(a)–(c) demonstrate that when the ER goes up (task radius increases, delay tolerance increases, *MDR* decreases), the task coverage increases for all frameworks. CROWDBIND is better than other frameworks because it can capture more dynamic users. The advantage of CROWDBIND relative to the other protocols decreases with the increase of ER as it becomes easier for all protocols to achieve near-perfect coverage.

Scheduling Fairness

Figure 9(a)–(c) show the results for scheduling fairness achieved by all four protocols with varying task radii, delay tolerances, and *MDR*. Fairness is quantified with the Gini coefficient [26], which we use to measure the inequality in the total number of times each device is selected during the experiment. A lower Gini coefficient indicates greater degree of fairness. Our ability to distribute tasks fairly (as well as that of Sense-Aid) is positively correlated with ER. This ratio can increase either due to a decrease in the *MDR* or an increase in the number of available devices. CROWDBIND captures more available users by predicting and including devices that will enter the task region within the delay tolerance of the task.

Figure 9(a)–(c) demonstrates how the task radius, delay tolerance, and *MDR* affect scheduling fairness. Task radius and delay tolerance have positive correlation with fairness. As the *MDR* increases, fairness for all protocols converges. CrowdRecruiter has the greatest inequality because it selects a fixed set of users that are most likely to satisfy all the task instances. Sense-Aid and CROWDBIND have an advantage over other frameworks because they prefer to select devices that have been selected less often. CROWDBIND further improves on fairness because of its use of mobility prediction. The difference between CROWDBIND and Sense-Aid narrows as the ER increases because larger ER gives both frameworks to find users capable of satisfying the task instance and thus have a greater opportunity to distribute the task instances equally among this larger set of users.

Figure 9(d) summarizes the result of an experiment with 6 tasks, each of which has a radius of 1 *km*, requires a single user, and consists of 16 instances spread over the course of 8 hours. In task 5, CrowdRecruiter has the highest inequality because it only selects 2 of the 20 available users. One of the users stays in the task region for 13 instances while another is available for 5 instances including the 3 task instances in which the first user is not in the task region. This selection results in an extremely unfair distribution with a Gini coefficient of 0.922. While Sense-Aid performs better than most frameworks, it is unable to surpass CROWDBIND in fairness.

Energy Efficiency

Figure 10 shows the effect of the different protocols on the client energy efficiency. We used the AT&T Video Optimizer [16] tool to compare the energy consumption of different protocols when sending one data point to the MCS server. The average energy consumption for the selected devices for each framework was calculated as $E_d = \frac{E}{N_s} = \frac{1}{N_s}\sum_i N_i \cdot e$ where E is the total amount of energy for completing a crowdsensing task, N_s is the number of selected devices, N_i is the number of data points sent by device i, and e is the energy consumption per data point sent by a device. Regarding the total energy E for the different frameworks, the following relation holds: $E_{\text{CROWDBIND}} = E_{Sense-Aid} < E_{CrowdRecruiter} < E_{PCS} < E_{Periodic}$. However, the energy efficiency metric exposes further differences.

In Figure 10(a), we see that with an increase in the task radius, both CROWDBIND and Sense-Aid see a decrease in the average energy consumption E_d. This is the result of a larger number of available users which increases the denominator N_s because both of these protocols consider fairness. Still, CROWDBIND wins over Sense-Aid because its N_s term is greater—a fact stemming from our two design choices:

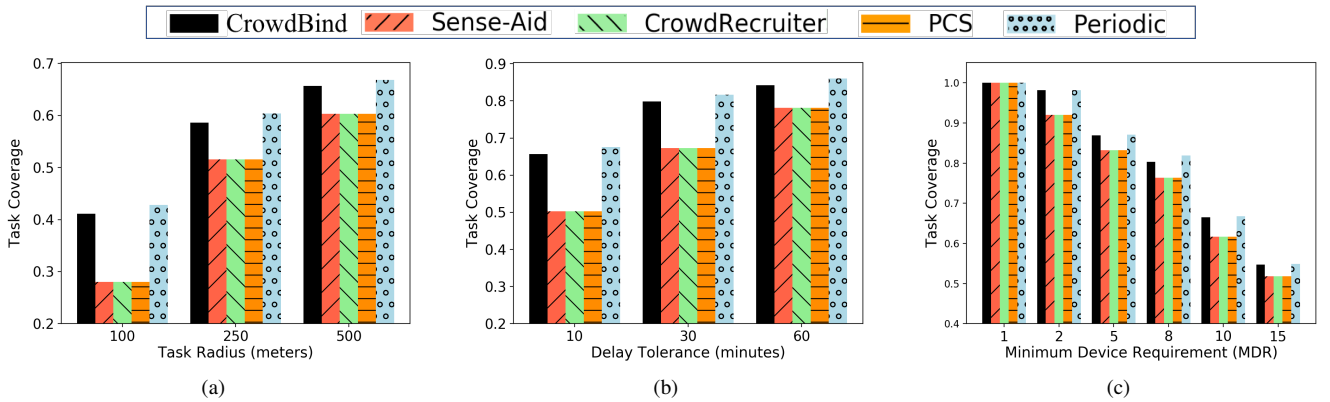


Figure 8: Task coverage in the user study with changes in (a) task radius, (b) delay tolerance and (c) minimum device requirement. CROWDBIND achieves near ideal task coverage for all experimental conditions (*i.e.,* same as Periodic).

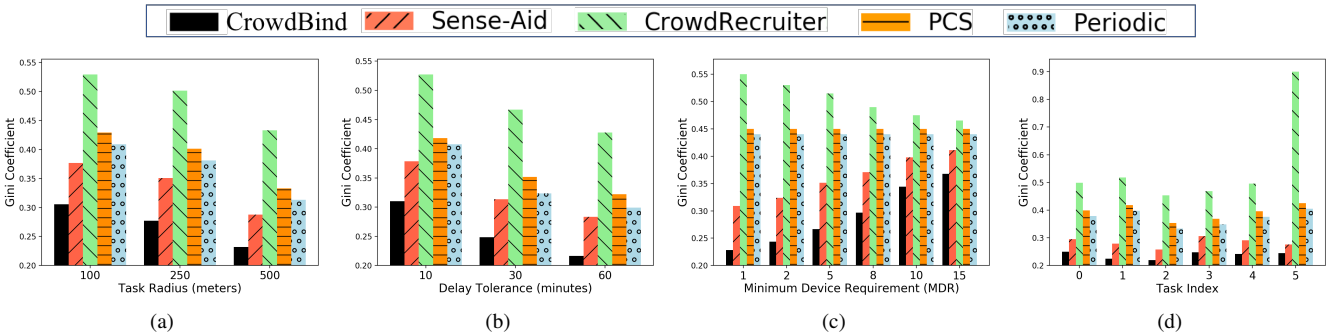


Figure 9: Scheduling fairness in the user study for varying (a) task radius, (b) delay tolerance, (c) minimum device requirement, and (d) an experiment with tasks requiring only one device within a 1 *km* radius. CROWDBIND achieves the highest fairness among the client devices.

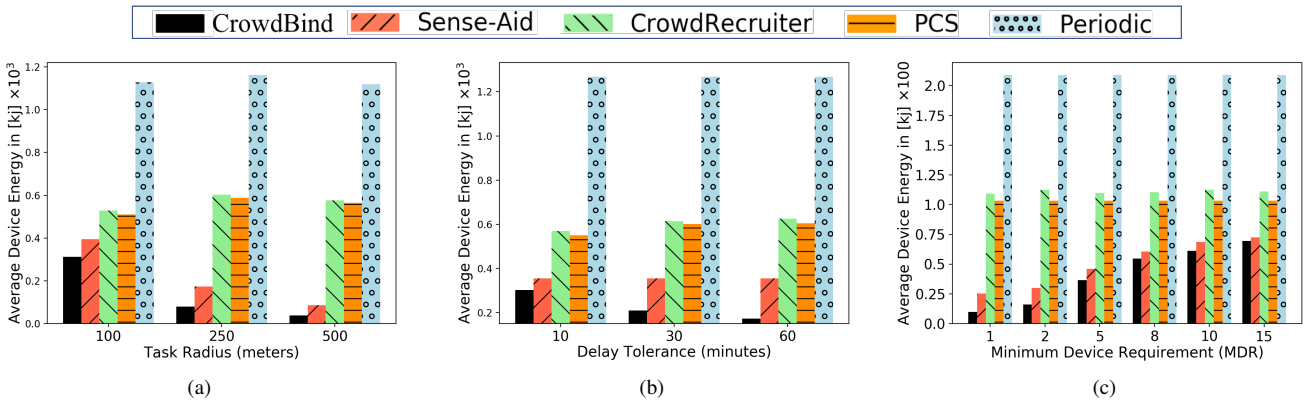
it has the ability to predict the future location of users *and* it is able to consider all the devices within this lookahead window for scheduling. Thus, CROWDBIND anticipates if a user will be within the task radius and the time window required by the task instance. Both Periodic and PCS schedule tasks to all devices capable of satisfying them and see high average energy usage as a result. CrowdRecruiter also underperforms because it schedules tasks to a minimal number of users rather than spreading out the tasks equally among available users.

When delay tolerance increases as in Figure 10(b), CROWDBIND has more users to select and therefore the energy efficiency is better. The other protocols stay relatively the same: the delay tolerance will not affect the average energy cost per device because they do not use the degree of freedom enabled by higher delay tolerance in their scheduling schemes. Sense-Aid, despite having some scheduling intelligence, chooses a device whenever the device can cover an instance. This means that a longer delay tolerance only gives it more chances to cover the task instance, but will not benefit the average energy efficiency. Periodic and PCS will schedule whenever there are users in the region. Longer delay tolerance will cause them to choose more users but the average energy will not change much. CrowdRecruiter picks the optimal k users independent of the delay tolerance and hence, its performance is not affected by the delay tolerance.

Figure 10(c) shows the impact of minimum device requirement (*MDR*) on E_d. CROWDBIND has the lowest energy cost for the entire range of *MDR* values and its benefit compared to Sense-Aid is 7.1%–66% and compared to CrowdRecruiter is 43%–95%. As *MDR* goes up, the relative benefit due to CROWDBIND goes down. PCS and Periodic do not consider *MDR* and use all the available devices. With CrowdRecruiter, the number of available devices (15 for our experimental setup) is always $\geq$ *MDR* and hence, a device is never called upon to serve multiple task instances, resulting in a flat curve. For CROWDBIND and Sense-Aid, as the *MDR* increases, ER decreases and therefore the energy cost goes up. To expand on this, with increasing *MDR*, the same device may be called upon to serve multiple task instances causing an increase in energy.

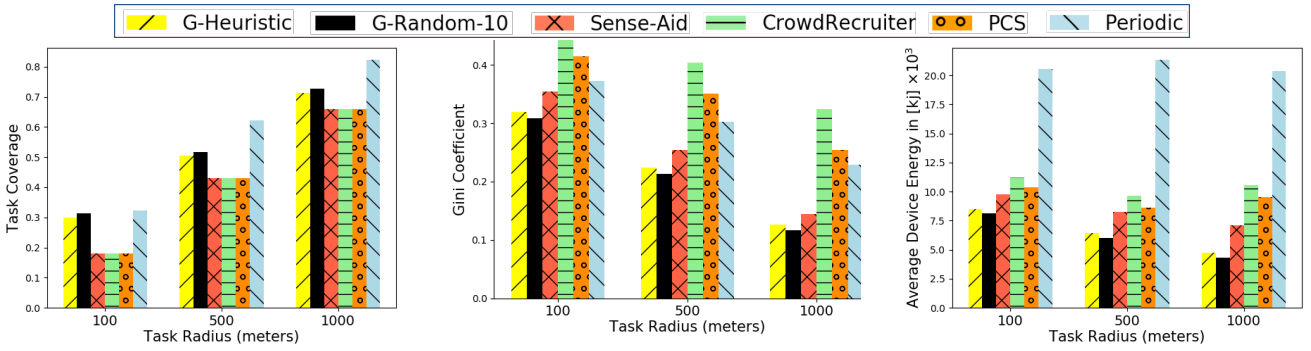
Part C: Simulation: CROWDBIND in Large Population

In this section, we evaluate the scalability of the two greedy algorithms (G-Random-10 and G-Heuristic) when applied to a large trace dataset. We compare them with the four prior works as those in Part B. Gowalla dataset [17] is used in this evaluation. Gowalla dataset has around 90K checked-in users. The trace for each user is an array of <*user id, timestamp, latitude, longitude, location id*>. The dataset starts from Feb. 2009 to Oct. 2010. We randomly choose one-month trace and iterate through all the location points to generate tasks. If a location point is outside of the circular area with R-meter radius of any existing tasks, a new task is generated. Otherwise, that location point is ignored. There are 112,920 tasks generated when task radius is 500 meters



(a) (b) (c)

Figure 10: Average energy consumption in the user study on selected devices with changes in (a) task radius, (b) delay tolerance, and (c) minimum device requirement. CROWDBIND achieves the highest energy efficiency due to the selection of the largest number of devices for the crowdsensing tasks.



(a) (b) (c)

Figure 11: Results from Gowalla dataset simulation with the change of task radius. (a) Both greedy algorithms achieve near ideal task coverage; (b) G-Random-10 achieves the best scheduling fairness among devices and G-Heuristic is close to the performance of G-Random-10; (c) G-Random-10 achieves the best energy efficiency due to the selection of the largest number of devices for the crowdsensing tasks. G-Heuristic is slightly higher than G-Random-10 but is still better than the other baseline solutions.

while that number is 79,500 meters when task radius is 1,000 meters. The initial usages of all users are zero. The default values are *task radius = 1000 meters, MDR = 4, MCR = 80%, delay tolerance = 20 minutes, periodicity = 20 minutes, and the duration of a single task = 1 hour*. Since the battery information is not included in the dataset, we assume all users have sufficient battery levels. With the change of task radius, Figures 11a, 11b, 11c shows the corresponding change of task coverage, scheduling fairness, and average energy cost. When calculating the average device energy, because different protocols have different task coverage, we only consider the commonly satisfied task instances to keep the total number of satisfied task instances the same. As we can see, the results are consistent with those in Part B. This evaluation shows that the insights from the user study carry over to a completely distinct and larger trace. We also ran experiments with the change of delay tolerance and MDR but they tell the same story as in user study and so we remove them for saving space.

5 Related Work

MCS task allocation is a popular topic in the literature. The works in [5, 12, 22] are all multi-task-oriented task allocation mechanisms with different optimization goals such as minimal travel distance, minimal incentive cost, and maximum task coverage. In the recent work, *PSTasker* [20] studies a heterogeneous task allocation problem in participatory MCS considering several different participant factors since participatory MCS is more dependant to participants. *CrowdTasker* and *CrowdMind* [24, 25] select a minimal subset of users to reduce the incentive cost. These frameworks predict when phone calls will occur and send data concurrently to save energy. CrowdWatch [8] is a distributed energy saving technique. The *Probabilistic Registration* framework [6] uses a mobility-aware approach to find a minimal subset of users that are expected to cover a certain ratio of the task instances. Chen *et al.* [1] study the task allocation considering the uncertainty in user movement in participatory MCS. Compared to the task allocation works above, CROWDBIND focuses on the optimization of the task scheduling fairness in opportunistic sensing and since the cost in our model is the energy consumption, we try to avoid repeatedly using the same user for different task instances in order to reduce the energy cost while some of the works above try to select the same user for different task instances in order to reduce the incentive cost. Ni *et al.* [13] propose that scheduling fairness is important in vehicle crowdsensing in order to avoid drivers as well as customes being disappointed. Liu *et al.* [11] study the social fairness in spatial crowdsourcing among workers.

Periodic, PCS, and Probabilistic Registration frameworks are *device*-centric in the sense that the devices, after re-

ceiving the tasks from the server, are responsible for determining when to collect and when to transmit the data. CrowdRecruiter, CrowdMind, and CrowdTasker are *network*-centric because the server is responsible for scheduling the data collection among available devices, which are passive entities providing data according to a server-determined schedule. Sense-Aid and this work combine these two approaches. However, Sense-Aid, relies on repeated querying of user location and does not perform any estimation of device availability over a window of time.

6 Discussion

The pre-knowledge of user location is helpful for MCS task scheduling. It will be significantly beneficial to integrate CROWDBIND with the cellular infrastructure. The location information at the granularity of base station is already available. Besides, the integration resolves users' privacy concern because the location data can stay with the cellular provider without sharing to third party.

7 Conclusion

In this paper, we uncover that the population values fairness in the domain of MCS. Our solution CROWDBIND formulates an objective function to maximize fairness, while not degrading the task coverage. We leverage the observation that MCS tasks are typically delay-tolerant and design a lookahead protocol that considers the allocation of task instances within the window and predicts the mobility of users in that lookahead window. CROWDBIND is able to achieve high fairness without impacting task coverage or energy efficiency. We show that CROWDBIND is able to satisfy more number of tasks (18.3% to 91.4% — range is across the 4 prior protocols) and distributing the tasks more fairly among available users (11.6% to 71.2%). Our simulation with Gowalla dataset shows the relative advantage of CROWDBIND is maintained.

8 References

[1] S.-F. Cheng, C. Chen, T. Kandappu, H. C. Lau, A. Misra, N. Jaiman, R. Tandriansyah, and D. Koh. Scalable urban mobile crowdsourcing: Handling uncertainty in worker movement. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 9(3):26, 2018.

[2] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE Communications Magazine*, 49(11), 2011.

[3] N. Gohring. App feeds scientists atmospheric data from thousands of smartphones., 2013.

[4] J. Goncalves, S. Hosio, N. Van Berkel, F. Ahmed, and V. Kostakos. Crowdpickup: Crowdsourcing task pickup in the wild. volume 1, page 51. ACM, 2017.

[5] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han. Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems. *IEEE Transactions on Human-Machine Systems*, 47(3):392–403, 2017.

[6] S. Hachem, A. Pathak, and V. Issarny. Probabilistic registration for large-scale mobile participatory sensing. In *Pervasive Computing and Communications (PerCom)*, pages 132–140. IEEE, 2013.

[7] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *Computer Communications (INFOCOM)*, pages 2254–2262. IEEE, 2015.

[8] R. Kravets, H. Alkaff, A. Campbell, K. Karahalios, and K. Nahrstedt. Crowdwatch: enabling in-network crowd-sourcing. In *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*, pages 57–62. ACM, 2013.

[9] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *International Conference on Ubiquitous Computing*, pages 243–260. Springer, 2006.

[10] N. D. Lane, Y. Chon, L. Zhou, Y. Zhang, F. Li, D. Kim, G. Ding, F. Zhao, and H. Cha. Piggyback crowdsensing (pcs): energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (Sensys)*, pages 1–14. ACM, 2013.

[11] Q. Liu, T. Abdessalem, H. Wu, Z. Yuan, and S. Bressan. Cost minimization and social fairness for spatial crowdsourcing tasks. In *International Conference on Database Systems for Advanced Applications*, pages 3–17. Springer, 2016.

[12] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, and D. Zhang. Taskme: multi-task allocation in mobile crowd sensing. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 403–414. ACM, 2016.

[13] J. Ni, A. Zhang, X. Lin, and X. S. Shen. Security, privacy, and fairness in fog-based vehicular crowdsensing. *IEEE Communications Magazine*, 55(6):146–152, 2017.

[14] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. S. Shen. Providing task allocation and secure deduplication for mobile crowdsensing via fog computing. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[15] L. Pu, X. Chen, J. Xu, and X. Fu. Crowdlet: Optimal worker recruitment for self-organized mobile crowdsourcing. *Network*, 4:5, 2016.

[16] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 137–150. ACM, 2010.

[17] S. Scellato, A. Noulas, and C. Mascolo. Exploiting place features in link prediction on location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1046–1054, New York, NY, USA, 2011. ACM.

[18] H. Si, Y. Wang, J. Yuan, and X. Shan. Mobility prediction in cellular network using hidden markov model. In *Consumer Communications and Networking Conference (CCNC)*, pages 1–5. IEEE, 2010.

[19] E. Thuillier, L. Moalic, S. Lamrous, and A. Caminada. Clustering weekly patterns of human mobility through mobile phone data. *IEEE Transactions on Mobile Computing*, 17(4):817–830, 2018.

[20] J. Wang, F. Wang, Y. Wang, D. Zhang, B. Y. Lim, and L. Wang. Allocating heterogeneous tasks in participatory sensing with diverse participant-side factors. *IEEE Transactions on Mobile Computing*, 2018.

[21] J. Wang, L. Wang, Y. Wang, D. Zhang, and L. Kong. Task allocation in mobile crowd sensing: State of the art and future opportunities. *arXiv preprint arXiv:1805.08418*, 2018.

[22] J. Wang, Y. Wang, D. Zhang, F. Wang, H. Xiong, C. Chen, Q. Lv, and Z. Qiu. Multi-task allocation in mobile crowd sensing with individual task quality assurance. *IEEE Transactions on Mobile Computing*, 2018.

[23] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren. Real-time and spatio-temporal crowd-sourced social network data publishing with differential privacy. *IEEE Transactions on Dependable and Secure Computing*, 15(4):591–606, 2018.

[24] H. Xiong, D. Zhang, G. Chen, L. Wang, and V. Gauthier. Crowdtasker: Maximizing coverage quality in piggyback crowdsensing under budget constraint. In *Pervasive Computing and Communications (PerCom)*, pages 55–62. IEEE, 2015.

[25] H. Xiong, D. Zhang, Z. Guo, G. Chen, and L. E. Barnes. Near-optimal incentive allocation for piggyback crowdsensing. *IEEE Communications Magazine*, 55(6):120–125, 2017.

[26] S. Yitzhaki. Relative deprivation and the gini coefficient. *The quarterly journal of economics*, pages 321–324, 1979.

[27] D. Zhang, L. Wang, H. Xiong, and B. Guo. 4w1h in mobile crowd sensing. *IEEE Communications Magazine*, 52(8):42–48, 2014.

[28] D. Zhang, H. Xiong, L. Wang, and G. Chen. Crowdrecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 703–714. ACM, 2014.

[29] H. Zhang, S. Bagchi, and H. Wang. Integrity of data in a mobile crowdsensing campaign: A case study. In *Proceedings of the First ACM Workshop on Mobile Crowdsensing Systems and Applications*, pages 50–55. ACM, 2017.

[30] H. Zhang and M. A. Roth. One month mobilty trace. *https://github.com/LLADzhang/CrowdBind/blob/master/trace.npy.gz*, 2018.

[31] H. Zhang, N. Theera-Ampornpunt, H. Wang, S. Bagchi, and R. K. Panta. Sense-aid: A framework for enabling network as a service for participatory sensing. In *Middleware '17: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 68–80, New York, NY, USA, 2017. ACM.