Scalable Urban Mobile Crowdsourcing: Handling Uncertainty in Worker Movement

SHIH-FEN CHENG, CEN CHEN, THIVYA KANDAPPU, HOONG CHUIN LAU, ARCHAN MISRA, NIKITA JAIMAN, RANDY TANDRIANSYAH, and DESMOND KOH, Singapore Management University

In this work, we investigate effective ways of utilizing crowdworkers in providing various urban services. The task recommendation platform that we design can match tasks to crowdworkers based on workers' historical trajectories and time budget limits, thus making recommendations personal and efficient. One major challenging we manage to address is the handling of crowdworker's trajectory uncertainties. In this work, we explicitly allow multiple routine routes to be probabilistically associated with each worker. We formulate this problem as an integer linear program whose goal is to maximize the expected total utility achieved by all workers. We further exploit the separable structures of the formulation and apply the Lagrangian relaxation technique to scale up computation. Numerical experiments have been performed over the instances generated using the realistic public transit dataset in Singapore. The results show that we can find significantly better solutions than the deterministic formulation, and in most cases we can find solutions that are very close to the theoretical performance limit. To demonstrate the practicality of our approach, we deployed our recommendation engine to a campus-scale field trial, and we demonstrate that workers receiving our recommendations incur fewer detours and complete more tasks, and are more efficient against workers relying on their own planning (25% more for top workers who receive recommendations). This is achieved despite having highly uncertain worker trajectories. We also demonstrate how to further improve the robustness of the system by using a simple multi-coverage mechanism.

CCS Concepts: • **Human-centered computing → Mobile computing**;

Additional Key Words and Phrases: mobile crowdsourcing, spatial crowdsourcing, participatory sensing, uncertainty modeling, context-aware, empirical study, user behavior

ACM Reference format:
Shih-Fen Cheng, Cen Chen, Thivya Kandappu, Hoong Chuin Lau, Archan Misra, Nikita Jaiman, Randy Tandriansyah, and Desmond Koh. 0. Scalable Urban Mobile Crowdsourcing: Handling Uncertainty in Worker Movement. *ACM Trans. Intell. Syst. Technol.* 0, 0, Article 0 (0), 24 pages.
DOI: 0000001.0000001

1 INTRODUCTION

In the past few decades and the foreseeable future, an irreversible global trend is for the majority of mankind to move to cities. As more and more large cities emerge around the globe, it is becoming evident that providing a broad spectrum of urban services sustainably would be increasingly challenging [39]. A promising approach is to enlist the help from the urban dwellers to provide some of these services (either through voluntary contributions or monetary incentives). Examples of services that are already outsourced this way include citizen sensing (i.e., asking participants

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative, and partially supported by the Xerox Research Centre India.
Author's addresses: School of Information Systems, Singapore Management University.

to contribute sensor readings [14]; specific measurement examples include pollution [37], traffic congestion [34], and noise level [40]), price checks, store audits (e.g., checking shelves, store displays), logistics (package pickup and delivery) [2], crowd estimates (at movie theaters and food courts), or organizing volunteer network (either digitally [25] or physically), just to name a few.

Such outsourcing scheme can be generally regarded as the paradigm of *mobile crowdsourcing*, which is a rapidly growing extension to the traditional crowdsourcing paradigm. In the traditional crowdsourcing paradigm (introduced by Howe and Robinson in 2005 and elaborated in Howe [9]), task owners distribute tasks (or sub-parts, often referred to as "microtasks") via online platforms to a distributed pool of crowdworkers, who perform these tasks for small payoffs. Driven by platforms such as Amazon Mechanical Turk and CloudFactory, this paradigm has become an integral part of many business operations and processes. It has also spawned the emergent field of *human computation*, where AI researchers study issues related to coordination, incentive design, process design, task assignment optimization and even ethics.

Mobile crowdsourcing is conceptually identical to its predecessor, yet it is unique in the following two ways: (a) tasks having strong *location specificity* (i.e., the task requires a crowdworker to physically visit one or more specific locations) and (b) tasks principally requiring the use of mobile devices/smartphones at these locations. Although still in its infancy (it is made possible only by the rapid growth of smartphone ownership), the mobile crowdsourcing paradigm is already demonstrating great promise in reducing operational cost and response time (for task owners), it also enables better utilization of crowdworkers.

Most academic research and practical deployments of mobile crowdsourcing presently employ a *pull-based* model. In this model, each individual worker operates independently, searching through, and selecting from, the corpus of available tasks (often with built-in proximity filters that enable them to identify jobs that are close to their current locations). As pointed out by Musthag and Ganesan [20], such pull-based embodiments of mobile crowdsourcing, as in all other forms of crowdsourcing, suffer from the phenomenon of *super agents*; i.e., a small percentage of crowdworkers who perform the majority of tasks[1]. Such skew in task selection is undesirable, as many ordinary crowdworkers might drop out as a result of not having enough tasks, reducing the worker pool and thus the peak capacity of the mobile crowdsourcing platform. By examining empirical data, Musthag and Ganesan conclude that the major difference between ordinary worker agents and super agents is the latter's ability in planning better routes and choosing tasks that fit their routes best.

To help improve crowdworker productivity via improved task selection, we introduced the *TRACCS* [5] framework for *push-based* mobile crowdsourcing. In this approach, the crowdsourcing platform utilizes a forecast of the travel path of each potential crowdworker to centrally plan for task recommendation, taking into account each mobile crowdworker's likeliest daily *routine route*. Task recommendation then attempts to match workers to tasks on, or near to, their individual routes, thereby minimizing their detour. This proposed recommendation system not only enables more tasks to be completed with less time spent in *detours* but also distributes tasks more evenly, thus reducing the negative impacts of super agents. However, this prior work makes the simplifying assumption that each crowdworker has one and only one *deterministically-known* routine route.

In this paper, we extend *push-based* mobile crowdsourcing to incorporate a more realistic scenario, where an individual worker's movement trajectory has inherent uncertainty (as the worker may travel on different routes on different days). Such uncertainty presents a technical challenge to effective task recommendation under the push-based model: the recommendation strategy must still recommend individual workers (modeled as independent agents) tasks that are *likely* to lie

[1]Based on their study, 10% of most active users complete 80% of all completed tasks.

along their routine commuting routes, while accounting for and trying to mitigate the likelihood that their actual routes (on a specific day) might make it infeasible for them to perform one or more of the recommended tasks.

By studying this novel problem of *multi-agent task recommendation, under stochastic spatiotemporal uncertainty*, we make the following major contributions:

- First, we show that this task recommendation problem can be formulated as an integer linear programming model, assuming that each agent's list of possible routine trajectories is finite, and governed by a known probabilistic distribution. The robustness of this model is further enhanced by incorporating a *multi-coverage* constraint, whereby each task is required to be recommended to a pre-determined number of distinct agents.
- Second, we develop and evaluate a combination of heuristics to address this computationally intractable problem. More specifically, we exploit the problem's structure to decompose the problem into independent agent-specific routing subproblems (each of which can be solved very efficiently), where the global coupling constraint is relaxed using the Lagrangian relaxation framework. Applying this computationally tractable framework to representative city-scale commuting behaviors (where empirical data indicates an average of ~ 2 primary commuting routes per worker), we show that we can obtain significant improvement (around $8\% - 15\%$) over the deterministic model that assumes only a single possible route. Moreover, this framework's performance is almost identical to the cases where perfect information is available (i.e., where each agent's daily routine route realization is assumed to be exactly known in advance).
- Finally, we embed our developed heuristic solver within a real-world mobile crowdsourcing platform, deployed on our campus. Using real-world field trials, we demonstrate that our push-based approach that considers routine route uncertainties can indeed significantly improve crowdworker's efficiency, compared to a pull-based alternative. In particular, we observe that crowdworkers receiving our path-based recommendations incur fewer detours and complete more tasks; as a consequence, the top push-based workers achieve an efficiency (reward earned per unit detour) that is over 20% higher than comparable pull-based workers. In addition, our push-based workers are able to plan their task selection more in advance. Most notably, this efficiency improvement is achieved in spite of highly unpredictable routine routes (the top 5 routes capture only about 50% of the exhibited movement behaviors) traversed by students on campus. The multi-coverage mechanism is also tested, and we show that by raising coverage ratio from 1 to 4 (i.e., recommending a task to 4 distinct workers), we achieve 26% increase in task completion.

The rest of paper is organized as follows. In Section 2, we review all streams of related work. In Section 3, we describe the push-based recommendation model. In Section 4, we explain how the Lagrangian relaxation approach can be utilized to speed up the solution process. In Section 5, we present the numerical experiments demonstrating the effectiveness and efficiency of our approach. In Section 6, we discuss an implemented real-world field trial. Finally, we conclude the paper in Section 7.

2 RELATED WORK

Mobile Crowdsourcing: Recent approaches such as [1], [16], [15], and [10] have focused on a particular class of mobile crowdsourcing, called participatory sensing, with a goal of maximizing the number of assigned tasks based on agents' current locations. For example, Kazemi and Shahabi [16] developed a centralized allocation algorithm focused on maximizing an agent's set of allocated tasks, while satisfying a proximity constraint (which implied that some tasks could be potentially performed by multiple agents). Sadilek et al. [24] studied a general class of problems called crowdphysics where tasks requires people to collaborate in a synchronized space and time. They reduced the problem into a graph planning problem and proposed two methods: global coordination using shortest-path algorithm and opportunistic routing based on the ranking of the time-stamped locations.

Researchers have also investigated the impact of incentives on task assignment/selection behavior in mobile crowdsourcing. Rula et al. [23] investigated the relative impacts of two different incentive mechanisms—micro-payments vs. weighted lotteries, on the task acceptance and completion rates of mobile agents (crowdworkers). The MSensing approach by Yang et al. [36] focuses on incentive design for mobile crowdsourcing tasks, employing either a Stackelberg game model (for scenarios where the reward is specified by the platform) or an auction-based agent selection mechanism (where the reward is determined through competitive bidding) to model the dynamics between the pricing of individual tasks and the willingness of each agent (task worker) to perform that task. However, this approach either considers a single task in isolation or employs a cost function (for each agent) that fails to account for the regular movement trajectory of each agent. Other work on task assignment in mobile crowdsourcing addresses questions of reliability – for example, recently Boutsis and Kalogeraki [3] seek to maximize the reliability of crowdsourcing tasks by selecting agents, subject to a task completion latency constraint.

In all of these approaches, the feasibility of task assignment is defined by the task's proximity to the agent's current location. In contrast, our focus is on a coordinated mechanism for *task recommendation* that operates over a longer time horizon (e.g., a day). We also explicitly model the inherent stochasticity (uncertainty) in individual agent trajectories and seek to minimize their task-related detours.

Orienteering Problem & Multi-agent Task Allocation: The planning of an agent's route to perform tasks that maximizes utility can be seen as the Orienteering Problem (OP) or Prize-Collecting Traveling Salesman Problem. The goal is to find a route that maximizes utility while respecting budget constraints. This problem was originally defined in Tsiligirides [31] and motivated by the scheduling of a cross-country sport in which participants get rewards from visiting a predefined set of checkpoints. Since then, OP has been studied extensively by the Operations Research and AI communities and recently, Gunawan et al. [8] provided a recent survey of the OP variants, solution approaches and applications.

The closest variant of OP to our problem is the team orienteering problem (TOP), where a team of agents are scheduled to visit different locations to collect reward within a time budget [4]. However, to the best of our knowledge, none of the previous research to date on TOP actually considers the incorporation of agent-specific location information, namely the probability distribution of routine routes taken by agents. One other variant of OP is the Multi-Period Orienteering Problem with Multiple Time Windows (MuPOPTW, studied by Tricoire et al. [29]). In MuPOPTW, sales representatives need to visit a list of mandatory customers on a regular basis, while non-mandatory customers located nearby should also be considered and integrated into the current customer tours. While one may view the set of mandatory customers as the nodes on the routine routes and non-mandatory customers as the tasks in our problem, there is no predefined visiting sequence for the mandatory customers and the stochasticity of agent routine routes is not captured in that model.

Another branch of related literature is in the general area of multi-agent task allocation, which looks at task allocation in distributed environments. For example, the issue of task allocation has been studied extensively in the load balancing of distributed systems [11, 12]. When task execution can potentially fail, there is also past research on how task allocation mechanisms can be tweaked to generate robust outcomes [21].

Real-world Implementations: There are already several notable systems that have demonstrated values of mobile crowdsourcing. Systems such as mCrowd [35], Twitch crowdsourcing [32] and Slide-to-X [30] provide platforms to enable various crowdsourcing tasks. Commercially, the most visible examples are Uber-like services that provide a platform for willing "part-time drivers" to offer rides to passengers. Several other applications enable smart citizen monitoring by using crowdsourcing for, e.g., detecting potholes [18], mapping noise [22] and pollution [26] in urban areas, and monitoring road traffic [19, 28]. Several instances of paid mobile crowdsourcing start-ups have also emerged commercially including FieldAgent[2], GigWalk[3], and NeighborFavor[4]. They pay workers a few dollars for microtasks such as price checks, product placement checks in stores, and location-aware surveys. Our real-world deployed system focuses on two core problems of all these systems: (1) assigning or recommending location-specific tasks to workers, and (2) exploring the relationship between worker behavior (e.g., accept/reject a task, time taken to choose the right task) and attributes of the tasks (e.g., rewards and locations).

Task Acceptance & Completion Behavior: Limited studies have explored the relationship between task attributes and worker behavior, especially for the crowdsourced execution of location-dependent tasks. Wang et al. [33] studied the task completion times of online tasks (posted on Amazon Mechanical Turk), and established a power-law relationship between task completion times and task-related features, such as the type of the task, the task price and the day the task was posted. Alt et al. [1] used an independently developed mobile crowdsourcing platform to discover a variety of worker preferences, including preference for performing tasks before and after business hours or involving relatively simple chores (e.g., taking pictures). More recently, Thebault-Spieker et al. [27] conducted studies on the relationship between task pricing and location, at city-scale, and showed that workers preferred to perform tasks with lower detours and that were outside economically-disadvantaged areas.

In contrast to this body of academic work on task recommendation and experimental studies on city-scale worker behavior, our focus is to empirically investigate the human dynamics of mobile crowdsourcing in an urban campus-like setting, and to uncover key behavioral differences arising from the use of push vs. pull models for task recommendation and selection.

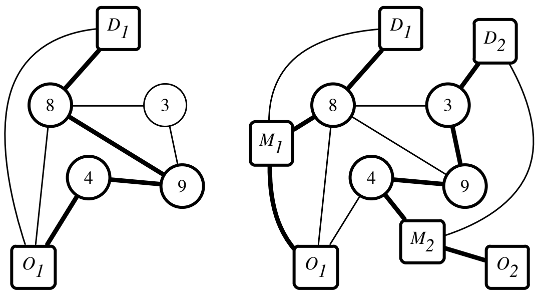
3 THE MODEL

As mentioned in Section 1, we are interested in creating models for recommending tasks to mobile crowdworkers with known routine route distributions, so as to maximize the expected total rewards collected by all agents. The major parameters guiding the task recommendation for each individual agent are the maximal amount of detour time allowed, the routine route that specifies the list of nodes each agent needs to traverse in order, and the probability distribution over a collection of routine routes (if this agent has multiple routine routes).

Mathematically speaking, this can be viewed as a specialized routing problem with time budget constraint and routine route requirement, and can be modeled as a variant of the well-known *orienteering problem* (OP). The classical OP can be seen in Figure 1a. In the classical OP, usually the requirement is for an agent to begin his trip from a given origin O_1, and to end at a given destination D_1. An agent can visit any node in-between O_1 and D_1 and incur corresponding link travel costs; yet he cannot exhaust his time budget before reaching D_1. The objective is for an

[2]http://www.fieldagent.net
[3]http://www.gigwalk.com
[4]http://www.favordelivery.com

individual agent to maximize value collected at visited nodes (each node comes with a different value). The OP variant for the mobile crowdsourcing problem with deterministic routine route (for easy explanation) can be seen in Figure 1b. There are two major differences when compared to the classical OP in Figure 1a: 1) there are multiple agents involved, and the planner aims to optimize the sum of all agent's values; and 2) for each agent, a routine route is specified, and its partial order needs to be followed. The routine routes in Figure 1b are characterized by O_i, D_i, and M_i, which represent origin, destination, and intermediate stop respectively (e.g., in Figure 1b, agent 1 needs to follow $O_1 - M_1 - D_1$, while agent 2 needs to follow $O_2 - M_2 - D_2$). Although there is only one intermediate stop for agents 1 and 2, in general, it is possible to have multiple intermediate stops in an agent's routine route. As in the classical orienteering problem formulation, we assume that all nodes can be visited at most once.



(a) Classical OP. (b) OP variant for mobile crowdsourcing.

Fig. 1. Illustrations of OP variants.

To model uncertainty on agent's routine route, we assume that each agent may take one of multiple routes with a known probability distribution. The objective is to optimize the sum of each agent's *expected* collected values. For the rest of the section, we will introduce an integer linear programming (ILP) formulation for the problem of mobile crowdsourcing with routine route uncertainty.

Note that our solution approach generates task recommendations for agents prior to the realization of actual routes taken. Such approach requires no agent inputs, and thus can reduce agent's cognitive burden in picking tasks; such approach is thus superior to the alternative where an agent has to first specify a deterministic route, before receiving task recommendation (this was the model in Chen et al. [5]).

3.1 Notations

We follow the classical notations used in formulating routing and orienteering problems. The list of notations can be found in Table 1.

Besides these notations, we have the following four sets of decision variables:

- $y_{ik} \in \{0, 1\}$: set to 1 when task i is recommended to agent k.
- $x^m_{ijk} \in \{0, 1\}$: set to 1 when agent k moves from nodes i to j when the realized routine route is m.
- $u^m_{ik} \in \{0, \dots, N\}$: indicates the visit order of node i for agent k, when the realized routine route is m.

Table 1. Notations.

N	the set of locations which comprise both the agent routine nodes and the task nodes
N_t	the set of task nodes
t_{ij}	$i, j \in N$ and $i \neq j$, travel time to move from i to j
K	the set of agents
M_k	the set of agent k's routine routes
β_k^m	$m \in M_k$, the probability that agent k would use route m
R_k^m	the collection of all nodes in agent k's route m
o_k^m	the origin of agent k's route m
d_k^m	the destination of agent k's route m
p_{ik}^m	the visit order for node $i \in R_k^m$
s_i	$i \in N_t$, task i's reward
e_i	$i \in N_t$, task i's required execution time
b_k^m	the upper bound of the total detour time along route m for agent k

- $z_{ik}^m \in \{0, 1\}$: set to 1 when agent k fails to complete recommended task i, when the realized routine route is m; if task i is not recommended to agent k in the first place, z_{ik}^m is set to 0 for all $m \in M_k$.

Note that the first set of decision variables, y_{ik}, is planner's recommendation decision; while the rest of decision variables are used for evaluating the outcome of the recommendation under different routine route realizations for each agent.

3.2 Integer Linear Programming Formulation

The objective of the ILP formulation is to maximize expected total rewards earned by all agents, considering uncertainties over their routine routes. The quantity $y_{ik}(1 - \sum_{m \in M_k} \beta_k^m \cdot z_{ik}^m)$ refers to the probability that a task i can be finished, if it is recommended to agent k (i.e., if y_{ik} is set to 1). Hence, the objective is to:

$$\max \sum_{i \in N_t} s_i \sum_{k \in K} y_{ik} \left(1 - \sum_{m \in M_k} \beta_k^m \cdot z_{ik}^m \right). \tag{1}$$

The above objective function is not linear, and in the next subsection, we will propose a linearized model.

Constraint (2) ensures that each task is recommended to at most one agent.

$$\sum_{k \in K} y_{ik} \leqslant 1, \qquad \forall i \in N_t. \tag{2}$$

All other constraints are at agent-route level (k, m), i.e., for each agent, $k \in K$, and for each of his routine route realization, $m \in M_k$, the same set of constraints applies. These constraints are explained as follows.

The first group of constraints ensures that flows are consistent at all nodes. In particular, (3) specifies that inflow and outflow at any node d must be balanced (except for the origin and destination nodes). (4) specifies that all routine nodes must be visited exactly once, while all other nodes can be visited by at most once.

$$\sum_{i \in N} x^m_{idk} = \sum_{j \in N} x^m_{djk}, \qquad \forall d \in N \backslash \{o^m_k, d^m_k\}, \tag{3}$$

$$\begin{cases} \sum_{j \in N} x^m_{djk} \leqslant 1, & \forall d \in N \backslash R^m_k, \\ \sum_{j \in N} x^m_{djk} = 1, & \forall d \in R^m_k \backslash \{d^m_k\}, \\ \sum_{i \in N} x^m_{idk} = 1, & d = d^m_k. \end{cases} \tag{4}$$

The time budget constraint for each routine route is enforced in (5).

$$\sum_{i \in N} \sum_{j \in N} x^m_{ijk} \left(t_{ij} + e_i\right) \leqslant b^m_k. \tag{5}$$

The following group of constraints produces visit orders (u^m_{ik}) from flows (x^m_{ijk}), and ensures that all nodes in the routine route m are visited according to the given sequence p^m_{nk}. In particular, (6) states that the visit order of the origin node must be 1; (7) states that if j is visited immediately after i (i.e., $x^m_{ijk} = 1$), the visit order of j should be *at least* 1 more than the visit order of i (i.e., $u^m_{jk} \geq (u^m_{ik} + 1)$).

$$u^m_{ik} = 1, \qquad i = o^m_k, \tag{6}$$

$$(u^m_{ik} + 1) - u^m_{jk} \leqslant N \; (1 - x^m_{ijk}), \qquad \forall i, j \in N. \tag{7}$$

(8) states that the partial order between any pair of nodes in the routine route must be preserved.

$$u^m_{ik} - u^m_{jk} \geqslant p^m_{ik} - p^m_{jk}, \qquad \forall i, j \in R^m_k \; \& \; p^m_{ik} > p^m_{jk}. \tag{8}$$

Finally, (9) extracts whether a task node is bypassed ($z^m_{ik} = 1$) from the flow decision (x^m_{ijk}).

$$z^m_{ik} \geqslant 1 - \sum_{j \in N} x^m_{ijk}, \qquad \forall i \in N_t. \tag{9}$$

3.3 Linearization

As noted earlier, the objective function (1) is nonlinear, as it includes multiplicative terms composed of y_{ik} and z^m_{ik}, both of which are decision variables. We would linearize (1) by introducing δ^m_{ik} to replace z^m_{ik}:

- δ^m_{ik} is set to 1 if task i is recommended to agent k, yet cannot be completed when the realized routine route is m, and 0 otherwise. In other words, $\delta^m_{ik} = y_{ik} \cdot z^m_{ik}$.

With δ^m_{ik}, we can rewrite (1) as:

$$\max \sum_{i \in N_t} s_i \sum_{k \in K} \left(y_{ik} - \sum_{m \in M_k} \beta^m_k \cdot \delta^m_{ik} \right). \tag{10}$$

To characterize δ^m_{ik}, we would rewrite (9) for all (k, m) as:

$$\delta^m_{ik} \geqslant y_{ik} - \sum_{j \in N} x^m_{ijk}, \qquad \forall i \in N_t. \tag{11}$$

With the above modifications, the formulation is now linear, and can be solved as an ILP using standard commercial solvers such as CPLEX.

In this ILP formulation, we assume that each task is recommended to at most one agent. As agents might choose to ignore the task recommendation for various reasons, such single-agent recommendation scheme might not be robust in terms of task completion. In the following subsection, a straightforward extension is introduced to allow us to request for more than one agents to be considered for each task.

3.4 The Multi-Coverage Extension

As noted in the Introduction, an agent is more likely to accept a task if it lies along his routine route(s). In other words, the agent may choose to skip a recommended task if his realized routine route forbids him from completing the task (the computed recommendation plan maximizes expected reward, and some task recommendation might not be feasible for a particular route). To increase the robustness of the recommendation plan without making the model much more complicated, we require that each chosen task be recommended to at least η agents, where η is a given parameter. In such robust planning scheme, we will either recommend η agents to a task, or not assign the task if this is not possible. To achieve this, we would introduce two additional classes of decision variables:

- $v_i \in \{0, 1\}$: set to 1 if this task is recommended to η agents, 0 otherwise.
- $w_{ik}^m \in \{0, 1\}$: variable for linearization, representing $v_i \cdot \delta_{ik}^m$.

To realize this extension, the objective function (10) is rewritten as (12), and the assignment constraint (2) is rewritten as (13). (11) also needs to be included. To linearize $(v_i \cdot \delta_{ik}^m)$ with w_{ik}^m, which appears in the modified objective function (12), we need the set of constraints (14). Finally, to ensure that we only assign task i to agent k when task i can at least be completed by one of agent k's route, we also need to include constraint (15). In other words, for every tuple (i, k), if $y_{ik} = 1$, $\delta_{ik}^m = 0$ for at least one $m \in M_k$.

$$\max \sum_{i \in N_t} s_i \left(v_i - \sum_{k \in K} \sum_{m \in M_k} \frac{\beta_k^m \cdot w_{ik}^m}{\eta} \right), \tag{12}$$

$$\begin{cases} \sum_{k \in K} y_{ik} = \eta \cdot v_i, & \forall i \in N_t, \\ y_{ik} \leqslant v_i, & \forall i \in N_t,\ k \in K, \end{cases} \tag{13}$$

$$\begin{cases} w_{ik}^m \leqslant v_i, & \\ w_{ik}^m \leqslant \delta_{ik}^m, & \forall i \in N_t,\ k \in K,\ m \in M_k, \\ w_{ik}^m \geqslant v_i + \delta_{ik}^m - 1, & \end{cases} \tag{14}$$

$$\sum_{m \in M_k} \delta_{ik}^m \leq y_{ik}(|M_k| - 1), \forall i \in N_t,\ k \in K. \tag{15}$$

3.5 Scalability of the Model

Our proposed ILP model is a variant of the orienteering problem, which is a well-known NP-hard problem [7]. As such, although the above ILP model can be solved exactly using CPLEX, it will not scale to even moderate sizes. Yet we introduce this exact model for two purposes: 1) to provide the problem structure for use by our proposed Lagrangian relaxation heuristic (to be described in Section 4); and 2) to provide an experimental benchmark for the heuristic.

To make the ILP model more scalable, we propose the following performance-boosting preprocessing procedures on the data without affecting the optimality of the model.

- For each subproblem pair (k, m), we remove all tasks that cannot be reached within the given detour time. Therefore, instead of using global node sets N and N_t in (k, m)-level constraints, we would use (k, m)-specific node sets N_k^m and N_{tk}^m.

- To avoid having to solve shortest path routing problems explicitly in the ILP model, we pre-compute shortest path distances for all origins and destinations and store them in a N-by-N matrix. With this matrix, we can further eliminate all non-essential nodes from N_k^m. More specifically, only nodes along the routine route m and feasible task nodes N_{tk}^m need to be included in N_k^m.

With these preprocessing steps, we are now ready to formally introduce the Lagrangian relaxation heuristic for the ILP.

4 LAGRANGIAN RELAXATION

Lagrangian relaxation (LR) is a well-known approach for solving difficult combinatorial problems. The key idea of the LR approach, as pointed out by Fisher [6], is that "*many difficult integer problems can be viewed as easy problems complicated by a relatively small set of side constraints.*". By moving these constraints to the objective function (i.e., dualizing these side constraints), we can provide a better lower bound (for minimization problems). LR is particularly powerful if the optimization problem allows further decomposition after the dualization of difficult constraints.

In our ILP formulation, (11) is the set of complicating constraints that couples all subproblems together; therefore, we choose to dualize this set of constraints. Following the convention in the standard LR literature, we define $\boldsymbol{\lambda} = \{\ldots, \lambda_{ik}^m, \ldots\}$ to be the vector of Lagrangian multipliers associated with constraints in (11) (indexed as (k, m, i)), and convert the objective function to be a minimization function. By construction, the objective function value of the Lagrangian dual problem will always be greater than or equal to the original objective function value (the primal). If the LP algorithm converges, the primal and the dual values should be identical.

Denote the Lagrangian dual problem as $L(\boldsymbol{\lambda})$, with the following objective function:

$$\min \sum_{i \in N_t} s_i \left(\sum_{k \in K} \sum_{m \in M_k} \frac{\beta_k^m \cdot w_{ik}^m}{\eta} - v_i \right) + \sum_{i \in N_t} \sum_{k \in K} \sum_{m \in M_k} \lambda_{ik}^m \left(y_{ik} - \delta_{ik}^m - \sum_{j \in N} x_{ijk}^m \right). \tag{16}$$

All constraints except (11) remain the same. By observing the problem structure, we can further decompose $L(\boldsymbol{\lambda})$ into two different classes of subproblems. The first subproblem class is the *assignment subproblem*, which decides how tasks should be recommended to individual agents (involves only y_{ik}); the second subproblem class is the *routing subproblem*, which finds the exact node visit sequence for each (k, m) tuple (involves only x_{ijk}^m). As subproblems can be solved independently, this decomposition can further improve the efficiency of our LR algorithm.

There is only one assignment subproblem in our formulation, we denote it as $g(\boldsymbol{\lambda})$ and define it as:

$$\min \sum_{i \in N_t} s_i \left(\sum_{k \in K} \sum_{m \in M_k} \frac{\beta_k^m \cdot w_{ik}^m}{\eta} - v_i \right) + \sum_{i \in N_t} \sum_{k \in K} \sum_{m \in M_k} \lambda_{ik}^m \left(y_{ik} - \delta_{ik}^m \right), \tag{17}$$

$$\delta_{ik}^m \leqslant y_{ik}, \quad \forall i \in N_t, m \in M, k \in K, \tag{18}$$

together with constraints (13) and (14). (18) is included to further tighten this subproblem such that incompletion penalty will only be imposed if the task is recommended.

The routing subproblem is defined for each (k, m) tuple; therefore, the total number of routing subproblems is $\sum_{k \in K} |M_k|$. For each (k, m) tuple, let $f_k^m(\boldsymbol{\lambda}_k^m)$ be the corresponding routing subproblem, where

$$\boldsymbol{\lambda}_k^m = \{\ldots, \lambda_{i-1,k}^m, \lambda_{i,k}^m, \lambda_{i+1,k}^m, \ldots\},$$

and $f_k^m(\boldsymbol{\lambda}_k^m)$ is defined to optimize:

$$\min - \sum_{i \in N_t} \lambda_{ik}^m \sum_{j \in N} x_{ijk}^m, \tag{19}$$

subject to constraints (3) – (8).

4.1 Subgradient Descent Algorithm

The Lagrangian dual problem is solved by using the following subgradient descent algorithm:

ALGORITHM 4.1. *Subgradient Descent Algorithm*

- ***Initialization:*** *Set iteration $t = 1$. Set $\boldsymbol{\lambda}_t$ to be all zeros.*
- ***Iteration $t \geq 1$:***
  (1) ***Solving Duals:*** *Solve all subproblems given $\boldsymbol{\lambda}_t$. The dual objective function value is computed by summing up objective values from all subproblems, i.e.,*
  $$L(\boldsymbol{\lambda}_t) = g(\boldsymbol{\lambda}_t) + \sum_{k \in K} \sum_{m \in M_k} f_k^m(\boldsymbol{\lambda}_{k,t}^m).$$
  (2) ***Primal Extraction:*** *To obtain the primal objective function value corresponding to the current dual solution (which might not be feasible in the primal problem), we insert the routing dual solutions $\{x_{ijk}^m\}$ back into the original problem and execute the optimization. In other words, we optimize (12) subject to the same set of primal constraints (11) and (13) – (15). The primal value (used in the next step) is set to be the negative of the obtained objective value so that it is comparable with the dual value.*
  (3) ***Updating Lagrangian Multipliers:*** *Lagrangian multipliers are adjusted according to the degree of constraint violation; for violated (satisfied) constraints, the values of corresponding multipliers should be increased (decreased) accordingly. We adopt a well-known adaptive multiplier adjustment formula appeared in Fisher [6]:*
  $$\lambda_{ik,t+1}^m := \max\{0, \lambda_{ik,t}^m + \alpha_t(y_{ik} - \delta_{ik}^m - \sum_{j \in N} x_{ijk}^m)\},$$
  where α_t represents the update step size, and is defined to be adaptive to both the duality gap and the solution quality (the magnitude of constraint violation):
  $$\alpha_t = \frac{\mu_t \left(F^* - L(\boldsymbol{\lambda}_t)\right)}{\sum_{i \in N_t, k \in K, m \in M_k} \left(y_{ik} - \delta_{ik}^m - \sum_{j \in N} x_{ijk}^m\right)^2},$$
  where $(\{x_{ijk}^m\}, \{y_{ik}\}, \{\delta_{ik}^m\})$ are obtained by solving dual subproblems, F^* is the best primal value seen so far, and $L(\boldsymbol{\lambda}_t)$ represents dual value obtained in iteration t. μ_t is in the range of $(0, 2]$, and is defined as:
  $$\mu_t = \begin{cases} 2, & t = 0, \\ 0.5\ \mu_{t-1}, & \textit{if } L(\boldsymbol{\lambda}_t) \textit{ is non-improving for } \kappa \textit{ iterations.} \end{cases}$$
  (4) ***Termination:*** *The search process terminates either when the allocated time is up or the duality gap (i.e., $F^* - L(\boldsymbol{\lambda}_t)$) is below certain threshold. In our implementation, we observe both termination conditions.*

4.2 Speeding Up LR Implementation

The LR subproblems are independent of each other, as such, they can be solved in parallel. The performance of our LR heuristic thus depends on how subproblems are solved (slowest subproblem dictates overall solution time). The baseline approach for solving both the assignment and the routing subproblems is to optimize the mathematical programming models described in the earlier part of this section. In implementation, we utilize heuristics for both subproblems to boost performance.

The assignment subproblem can be solved exactly and efficiently using a greedy algorithm. This greedy algorithm is designed to exploit the fact that each and every task is considered independently, and all routing-related considerations (visit orders and detour times) are handled separately.

Algorithm 4.2. *A Greedy Algorithm for the Assignment Subproblem*

For each task i, compare the case where $v_i = 0$ (not performing the task) and $v_i = 1$ (performing the task). For $v_i = 0$, the objective value is trivially zero. For $v_i = 1$, compute the best achievable objective value as:

(1) *For $k \in K$, we can quantify agent k's contribution to the objective value by setting $y_{ik} = 1$ for components involving k in (17):*

$$\sum_{m \in M_k} \left(\lambda_{ik}^m + \delta_{ik}^m \left(\frac{s_i \cdot \beta_k^m}{\eta} - \lambda_{ik}^m \right) \right),$$

where we set $\delta_{ik}^m = 1$ if $\left(\frac{s_i \cdot \beta_k^m}{\eta} - \lambda_{ik}^m \right) < 0$ (because (17) is a minimization problem).

(2) *Sort all agents according to their contributions in ascending order. Choose the first η agents and set $y_{ik} = 1$ for these η agents. The objective value for task i can be computed by summing contributions for these η.*

If the objective value for $v_i = 1$ is negative, set $v_i = 1$, otherwise set $v_i = 0, y_{ik} = 0, \delta_{ik}^m = 0$ for all $k \in K, m \in M_k$.

Proposition 4.3. *Algorithm 4.2 always finds optimal solution.*

Proof. In Step (2) of Algorithm 4.2, if we replace any of the η agents, the objective value will only increase, leading to a suboptimal solution. Similarly, in Step (1) of Algorithm 4.2, the objective value will only increase if δ_{ik}^m is set differently.

Therefore, Algorithm 4.2 will always find the optimal assignment. □

The routing subproblem, on the other hand, cannot be solved so efficiently, and are the major performance bottleneck. To investigate the trade-off between the optimality and the time performance, we define the following two LR variants based on how the routing subproblems are solved:

- **LR-Exact**: Routing subproblems are solved exactly by pure enumeration. Pure enumeration is the preferred approach in most instances since with reasonable detour limit (say up to 30%), the number of feasible tasks will be small enough such that pure enumeration will outperform regular routing algorithm; a threshold can be set for the solver to switch to regular router if the number of feasible tasks is too large.
- **LR-Greedy**: Routing subproblems are solved using a simple greedy heuristic: an agent starts with the routine route m, and repeatedly try to evaluate the gain of inserting one of the remaining tasks into all potential slots; of all possible (task, slot) combinations, the best is chosen (thus greedy). There is no optimality guarantee, but it can solve routing subproblems very efficiently.

The **LR-Greedy** heuristic is very efficient, and can finish within $O(|R_k^m| \cdot |N_t|)$ even in the worst case (when there is no limit on worker's detour time). The **LR-Exact** algorithm, on the other hand, can perform very poorly as it simply enumerates all feasible task combinations. For smaller detour limits (e.g., the 30% as mentioned above), this might not be an issue. But for higher detour limits, the **LR-Exact** approach will experience exponential execution time growth in the number of tasks, and will not be scalable.

In the next section, we will empirically evaluate the effectiveness and efficiency of both approach, and quantify whether the quality and time trade-off of the **LR-Greedy** heuristic is worthwhile.

5 NUMERICAL EXPERIMENTS

The ILP model and the two LR heuristics have been introduced in Sections 3 and 4, with these, we have addressed the first major contribution of the paper: modeling task recommendation under route uncertainty using the ILP model and coming up with efficient heuristics. In this section, we investigate the computational properties of these approaches using synthetic datasets. These investigations will help us understand the performance trade-offs we would face when deploying our recommendation engine for real-world field trials (to be reported in the next section).

More specifically, we investigate the performance of our LR heuristics from the following two perspectives:

- **LR heuristics versus the exact approach**: To understand the trade-offs between computational efficiency and solution quality, we solve the same problem instances using both our LR heuristics and the ILP model (which returns the true optimum). This evaluation can only be conducted for small instances due to the complexity of solving the ILP model.
- **LR heuristics versus deterministic heuristics**: To understand the benefits of modeling route stochasticity explicitly, we compare the performance of the LR heuristics against two deterministic heuristic approaches. One is a *push-based* deterministic model proposed in [5]; the other is a *pull-based* proximity approach that emulates current best practices. We use a city-scale network topology to perform this comparison.

5.1 LR Heuristics versus the Exact Approach

The purpose of this evaluation is to compare LR-Exact and LR-Greedy to the exact ILP model, both in terms of solution quality and computational time. Test instances are generated randomly with parameters (K, N_t, N), where K refers to the number of agents, N_t refers to the number of task nodes, and N refers to the total number of nodes in the network. Each agent is assumed to have two routine route candidates, where all routes have 5 nodes and are selected with equal probability. The coordinates of all nodes are generated uniformly randomly on a grid network. The distance between all pairs of nodes are Euclidean distance.

Table 2. LR heuristics vs. ILP: on both quality and time.

(K, N_t, N)	ILP	LR-Exact		LR-Greedy	
	Time	Gap	Time	Gap	Time
(2,4,40)	0.8s	0%	0.09s	0%	0.05s
(4,8,80)	22.9s	0%	0.2s	4.12%	0.06s
(8,16,80)	6558s*	0.06%	14.8s	0.06%	0.14s

*: We cut off CPLEX solver as the optimality gap is only 0.06%.

Our evaluation is summarized in Table 2. The gap is percentage from the optimum obtained via solving the ILP model exactly. From these small testing instances, we can see that both LR heuristics can produce close-to-optimum solutions very quickly. But examining the results closer, we can see that the efficiency of LR-Exact and LR-Greedy can potentially be different by one to two orders of magnitude.

5.2 LR Heuristics versus Deterministic Heuristics

To empirically quantify the benefits of generating recommendations considering routine route uncertainties, we introduce the following two deterministic heuristic baselines to compare with:

- **Deterministic-ILS**: Following the deterministic model and the iterated local search (ILS) heuristic proposed by Chen et al. [5], we designate each agent's routine route to be the route with highest probability. We denoted this baseline as **DILS**.
- **Proximity-Based Approach**: This heuristic emulates how most pull-based mobile crowd-sourcing platforms work nowadays. At each decision epoch, the agents who are available will be given the opportunity to pick desired tasks based on proximity. We denoted this baseline as **Proximity**.

The network used in this evaluation is based on the actual public transit network in Singapore (4,296 nodes and 10,129 edges), which contains all stops from the metro and bus services. All-pair-shortest distance matrix is computed a priori. To reflect the heterogeneous travel patterns of agents, we include two types of agents: 80% of *normal agents* who commute back and forth between fixed locations (e.g., home and office), and 20% of *freelancers* whose routine routes have randomly chosen origin and destination nodes. For both agent types, the origin nodes are randomly picked from the non-central zones, while the end nodes are from the central zones (reflecting a commuting pattern from "home" to "office"). Two routes are constructed for each agent as follows: (i) the shortest path between the chosen origin and destination, and (ii) the path with the least number of stops. The probability distribution over agent k's two routes follows Bernoulli distribution with parameter α_k, where α_k is sampled uniformly from $(0, 1)$. Locations of tasks are generated according to the distribution (p_r, p_h), where p_r and p_h refer to the ratio of tasks in the non-central and the central zones. For the synthetic instances, half are generated with (p_r, p_h) equals: (60%, 40%), while the rest with (40%, 60%). Each task is associated with a fixed utility value of 100.

Table 3. LR heuristics vs. deterministic baselines.

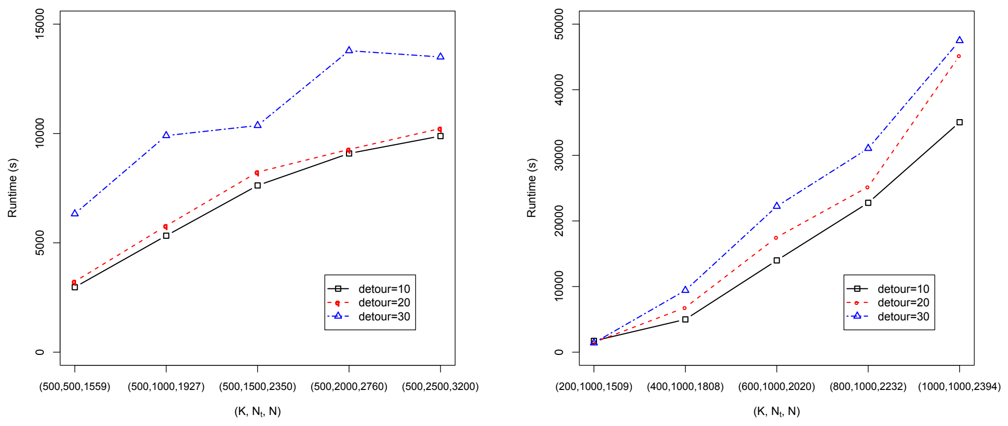
Detour	Estimated	LR-E	LR-G	DILS	Proximity
	Bound	Gap	Gap	Gap	Gap
10%	54.5%	-0.6%*	0.6%	13.2%	15.3%
20%	77.4%	-0.9%*	0.1%	10.4%	12.2%
30%	91.9%	0.1%	1.1%	7.1%	8.6%

The results for this section are summarized in Table 3, categorized using different detour limits for the tuple $(K, N_t, N) = (20, 30, 160)$, where 20 synthetic instances are randomly generated for every detour limit category using the above scheme. For each synthetic instance, all approaches are engaged to produce their respective task recommendation plans, which are then evaluated by 1000 routine route realizations. For each sampled route realization, we compute the percentages of tasks that can be accomplished for recommendation plans generated by all approaches (subject to the specified detour limit). For each approach, we then compute the average task completion percentage over 1000 route realizations and treat it as the performance of the approach.

To provide an estimated upper bound on the task completion percentage for each synthetic instance, we assume that the routine route realization is known during the evaluation phase and we solve the recommendation problem using DILS. This value is then used as the comparison baseline.

The reported gaps in Table 3 are all relative to this estimated bound (in other words, the smaller the better). Although the estimated bounds are obtained with perfect information on route realization, due to the heuristic nature of DILS, there are cases (denoted with *) where the LR-Exact approach outperforms the estimated bounds.

From Table 3 we can clearly see the advantage of considering route uncertainties: both LR-Exact and LR-Greedy are able to obtain significantly smaller gap compared to deterministic alternatives. The advantage of push-based approach (DILS) over pull-based approach (Proximity) is also significant, which is consistent with previously reported results. We repeat similar experiments with a wide range of tuples (K, N_t, N), and in all instances, LR-based approaches outperform deterministic alternatives, and the advantages of our LR heuristics increase further as we tighten detour limits.



(a) Runtime when N_t is increased.

(b) Runtime when K is increased.

Fig. 2. Runtimes of LR-Greedy in response to numbers of agents and tasks.

In terms of scalability, LR-Exact does not scale well. Without parallelization, LR-Exact takes more than 9 hours to solve (100, 200, 600) instances, which is of moderate size. LR-Greedy, on the other hand, is much more scalable. In Figure 2a, we fix K at 500 and increase N_t from 500 to 2500; in Figure 2b, we fix N_t at 1000 and increase K from 200 to 1000. From both figures, we empirically observe that runtime scales linearly in K and N_t.

6 *TA\$KER* : A REAL-WORLD MOBILE CROWDSOURCING PLATFORM

The applicability of our recommendation engine is put to test in a campus-scale mobile crowdsourcing platform called *TA\$Ker* . *TA\$Ker* is designed to allow students to register as crowdworkers, and a wide variety of tasks are designed and distributed via the platform. These tasks can be categorized into the following four categories:

(1) Discrete valued multiple choice: workers have to select from a predefined set of values. Example: *Is the male toilet at the second level of the Science building clean? Yes / No.*
(2) Counting-based: workers have to select from a predefined set of numerical values. Example: *How many people are queuing at the coffee shop? 0–10.*

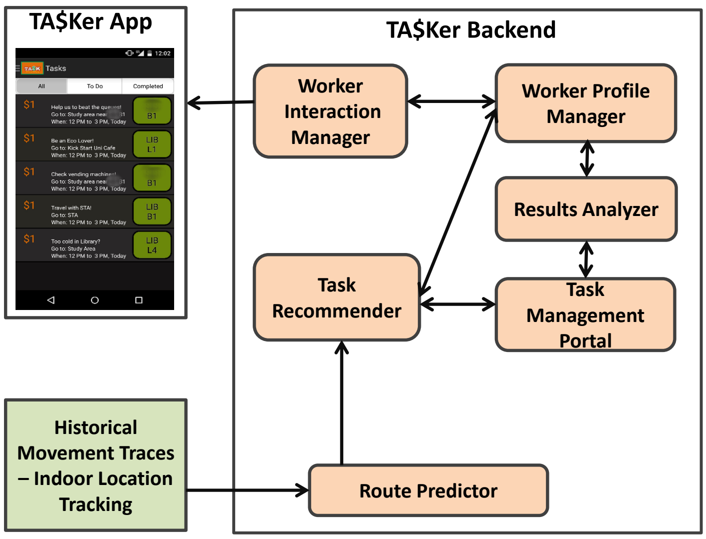


Fig. 3. Software architecture of the *TA$Ker* platform.

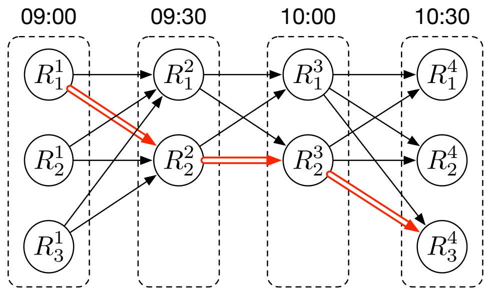
(3) Picture-based: workers are required to upload task-specific images using their smartphones. Example: *Snap a picture of the promotion sign currently hanging in front of the Men's Grooming section at the Watsons store on campus.*
(4) Free text-based: workers are to type in their answers as text. Example: *Tell us the price of AXE body spray sold at the Watsons store on campus.*

The *TA$Ker* platform is made possible by a number of front-end and back-end software components (see Figure 3). Task Recommender is the component that implements our LR heuristics, and the most important supporting component is the Route Predictor, which provides the probability distribution of routine routes for all participating students based on their historical movement traces. In the interest of this paper, we will only explain how the Route Predictor works (rest of the details of the *TA$Ker* platform can be found in Kandappu et al. [13]).

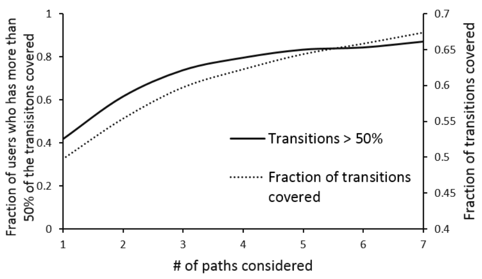
6.1 The Route Predictor

The movement models of all participating students are constructed using on-campus mobility traces collected in the months of August and September in 2015. Although trajectory mining from mobility traces has been studied extensively recently [38], the route prediction problem in our setting is relatively unique as our campus environment is mostly indoors. As such, we rely on Wi-Fi-based indoor localization techniques to track all student participants.

The mobility traces data consists of a series of tuples of the form <*participantID*, *locationID*, *timestamp*>, where the *participantID* is an anonymized ID representing a hash of the Wi-Fi MAC identifier of a Wi-Fi connected device and *locationID* refers to a location coordinate (successive coordinates are separated by a distance of 3 meters) in one of five academic buildings on our campus and a publicly accessible underground concourse connecting these buildings. The location data is obtained via a server-side Wi-Fi fingerprinting technique (employing the same location tracking approach as the one studied by Khan et al. [17]), and provides a median accuracy of 6-8 meters;



(a) Transition graph.



(b) Prediction accuracy.

Fig. 4. Figures depicting (a) a sample transition graph and (b) prediction accuracy of the route generator.

due to limitations on the commercial Wi-Fi infrastructure, the location updates occur sporadically, with a mean interval of approximately 3 minutes.

With these mobility traces, we create the following three-stage process to transform the raw data into routes:

(1) **Extract Reference Location:** We first sort each user's raw mobility data chronologically and calculate the amount of time spent in each location within every non-overlapping 30-minute time segment throughout the day. We then rank locations based on the amount of time spent and this process is repeated over all 30-minute time segments.
(2) **Construct Transition Graph:** Once reference locations are extracted, a transition graph is constructed to find out all the possible routes a user can take within a time window (e.g., for 9am − 12pm), and attach each route with a corresponding probability. In the graph, each node R_i^t represents the reference location i in time segment t, and a directed edge $e_{i,j}^{t,t+1}$ can only exist from time t to $(t+1)$, indicating a user's transition from location i to j between two consecutive time segments. Note that it is possible that $i = j$, which implies that a user dwells at the same location. Each edge comes with a probability p, which is estimated from the raw data, and is essentially the normalized dwell time over the entire set of this user's destinations in time $(t+1)$.
(3) **Generate Route Prediction:** Using the transition graph, the likelihood of each route can be calculated by multiplying all edge probabilities along this route. To focus on only the most probable routes, only top k routes are chosen per user. Note that these identified routes contain only "reference locations", which might be some distance apart. The real walk paths are generated by applying standard shortest path algorithm for all connected reference location pairs. The value of k in our experiments is set to 5, as the top 5 routes can explain more than 50% of all transitions for 85% of users.

For example, consider a user u. To predict his route on a typical Monday between 9am and 11am, the algorithm considers the past mobility traces of user u on Mondays and finds reference locations in all the 30-minute time segments (four time segments for our example). The extracted reference location R_i^t is denoted as reference location i in the t^{th} time segment and the weight of R_i^t is defined as normalized stay time of location i over the total stay time in time segment t.

The transition graph is then generated (see Figure 4a), with the transition probability from node R_i^t to R_j^{t+1} defined as the weight of node R_j^{t+1}. Global route likelihood is then calculated as the product of weights of all edges traversed in the route. The best route (maximizing the global route likelihood) is graphed using red double line in Figure 4a.

6.1.1 Prediction Accuracy. The accuracy of our prediction is evaluated based on the mobility traces of students who installed our App. We split our data into two parts for training and testing. The training data comes from the last week of August and the first three weeks September, and the testing data comes from the last week of September. For each weekday (Monday to Friday), a transition graph is constructed for each user based on four weekdays in August/September. The generated predictions for each weekday in the last week of September is then compared against the ground truth. We observe that our prediction algorithm achieves 68% precision and 75% recall.

In Figure 4b, we plot how k's value affects the fraction of transitions covered on the per user basis (for the primary Y-axis, we plot the fraction of users who have more than 50% of the transition coverage over k). The same figure also depicts how the k value affects the transitions covered in total for the whole system (for the secondary Y-axis, we plot the fraction of total transition coverage of the system). The value of k in our experiments is set to 5, as the top 5 routes can explain more than 50% of all transitions for 85% of users.

6.2 User Study Details

Results presented here are obtained from a user study conducted with student participants on our university campus. All experimental studies are conducted with approval from our Institutional Review Board on campus. As part of the study, we recruited 160 students, who were briefed on the functionalities of the App (but not told about the push vs. pull modes of task recommendation). To protect privacy, we did not extract any personally identifiable information such as name, date of birth, age and contact details. We then asked the students to perform tasks using our App at various locations on the campus. The trials were conducted over a two-week period from September 23 – October 2, 2015. During the trial, each student was free to use the *TA$Ker* App to perform any task that was in her list of *Available Tasks*.

To study the relative efficacy of push vs. pull models, 160 students were randomly and equally divided into the "Push" class and the "Pull" class. For students belonging to the "Push" class, they were provided task recommendations tailored to their predicted routes. In contrast, for students in the "Pull" class, they were able to see the entire set of available tasks, and have to make their own task selection. The user study was governed by several important parameters:

- *Task Time Windows:* To accommodate student's typical daily schedule, each day was divided into three 3-hour time windows: (a) 9am – 12pm, (b) 12pm – 3pm, and (c) 3pm – 6pm. New tasks were loaded into the App at the beginning of every new time window (i.e., at 9am, 12pm, and 3pm). Any task not completed by the end of its time window was considered expired and was removed from the list of available tasks.
- *Reward per Task:* As our study did not focus on incentive design, we adopted a flat reward structure: every successfully completed task resulted in an earning of $1.
- *Maximal Tasks per Time Window:* Each student was allowed to perform at most 3 tasks per time window.
- *Varying η:* To investigate the effectiveness of the coverage ratio per task, we experiment with different η values ranging from 1 to 4.

6.3 Evaluating the Performance of the Recommendation Engine

It is natural to ask: *given the inherent location errors and movement uncertainties of workers on the campus, can our recommendation strategy generate useful task recommendations?* To establish this efficacy, we compute the accuracy of two distinct recommendation strategies: (a) our proposed strategy where the recommendations take into account each worker's individual predicted trajectories, versus (b) a *trajectory-oblivious* strategy, where each worker is first assigned one of five "representative trajectories" (each of which traverses all the floors of one of five campus buildings), and the centralized recommendation algorithm is then executed on these synthetic trajectories. The recommendation error is then computed in terms of the "detour distance" to a task's location, with this detour being defined as the *minimum distance to the task location, from all reference locations that the worker actually visits during the task's validity period.* For example, assume that the recommended task T_1 (with location l_1) is valid during (10:00am, 10:30am) and the observed series of stay locations during this period is $\{X, Y, Z\}$. The minimum needed detour is then computed as $\min\{d(X, l_1), d(Y, l_1), d(Z, l_1)\}$.

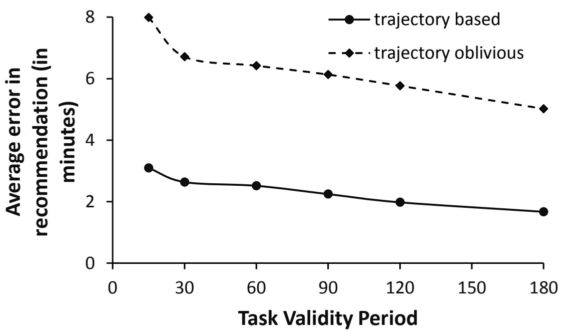


Fig. 5. Error in recommendations.

Figure 5 plots the average error (detour overheads across all users) versus task validity period for these two strategies. We can see that the average error of our recommender is around 2.5 times lower compared to the trajectory-oblivious approach. More specifically, for a 15-minute validity window, our recommendation error is less than 3 minutes (equivalent to travel times between floors of the same building), while the trajectory-oblivious approach has an error of over 8 minutes (equivalent to the time needed to visit a location three buildings away). We also can see that in general, as the task validity period increases, the error in recommendation decreases, as the additional slackness in time enables us to better predict a worker's precise trajectory.

To study the efficacy of η, the multi-coverage parameter, we considered data from another pilot (March-April 2015) for a 4-week period where 80 students signed up and completed 800 tasks. During this trial period, we incremented η each week from 1 to 4. We compute the task completion rate each week by calculating the ratio between the number of completed tasks in that week and the total number of tasks accepted (normalized over number of users participated in each week). We find that increasing η indeed helps to improve the task completion rate. With $\eta = 3$, the task completion rate is improved by 20%; when $\eta = 4$, the task completion rate is improved by 26%.

6.4 Key Insights on User Behaviors in Mobile Crowdsourcing

Our work represents perhaps the first attempt at tracking user behaviors in mobile crowdsourcing systems in a campus setting. In this section we report key findings of our trial analysis.

6.4.1 Super-Agent Phenomenon. While examining *TA\$Ker* user's behaviors, we observe an interesting trend – a relatively small number of users generate a disproportionally large fraction of task responses. Figure 6 shows that 30% of active agents are responsible for 70% of total tasks completed on the *TA$Ker* platform. The existence of such heavily skewed behavior makes it important to focus on this critical group of users since they play an important role in the overall dynamics of the system and contribute more value to the task owners. We refer to this top 30% of active agents as *super agents*.

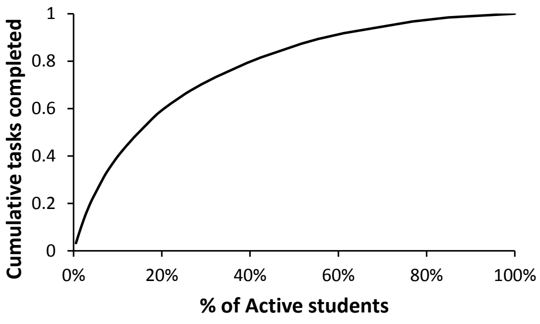


Fig. 6. Super-agent phenomenon.

6.4.2 Efficiency of Users. In this section, we illustrate how we measure a user's efficiency in performing mobile crowdsourcing tasks. In particular, we provide measurements on a user's *planning efforts* and *task performance efforts* and demonstrate how being in the push/pull classes and being super/normal agents would affect these performance measures. All results are summarized in Table 4. The definitions of all used metrics are explained below.

Detour: To compute detour efficiency, we need to estimate detours that are related to the performance of tasks. However, measuring the total time traveled by a user is not straightforward since we need to identify the neighboring stay locations (both prior to and after the task performance) in which the user stays for a significant amount of time (in our case, more than 4 minutes) to calculate additional time elapsed for him to reach his next location after deviating from his usual route to perform the chosen task.

Let the task location be denoted as Z, we analyze the location traces, and identify locations this user stayed at, for considerably longer time before and after going to Z. We denote the location this user stayed before Z as X, and the location after Z as Y. The detour time is then $(t_{X,Z} + t_{Z,Y}) - t_{X,Y}$, where $t_{X,Z}$ denotes the travel time to reach location Z from location X.

Figure 7 shows the histogram of the total detour made per user throughout our trial period. After averaging over the detour time over the number of completed tasks, we find that users from the push class incurred a detour of 5.2 minutes per task, which is 2.4 minutes shorter on average than users in the pull class. A t-test confirms that the push class indeed incurs statistically lesser detour time than the pull class with a p-value of 0.0016. In terms of labor supply, super agents contributed on average 45 minutes while normal agents contributed only 15 minutes on average.

To further study how the push and pull modes affect behaviors of super agents, we separately analyze behaviors of super agents and normal agents in both push and pull classes. In the push class, super agents incurred 40 minutes of detour while normal agents incurred only 12 minutes. The same trend is observed in the pull class – super agents made significantly more detour compared to normal users (62 and 18 minutes, respectively).

Detour Efficiency: We have seen that super agents are willing to contribute more detour time, but does the extended travel lead to more earning opportunities? We measure this by the *detour*

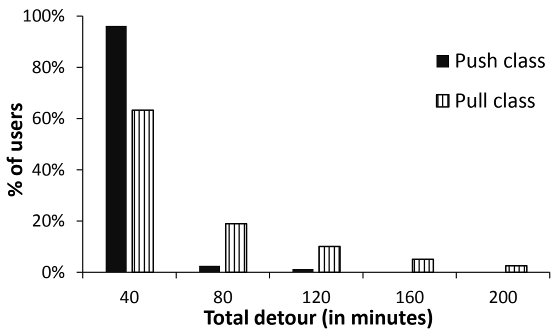
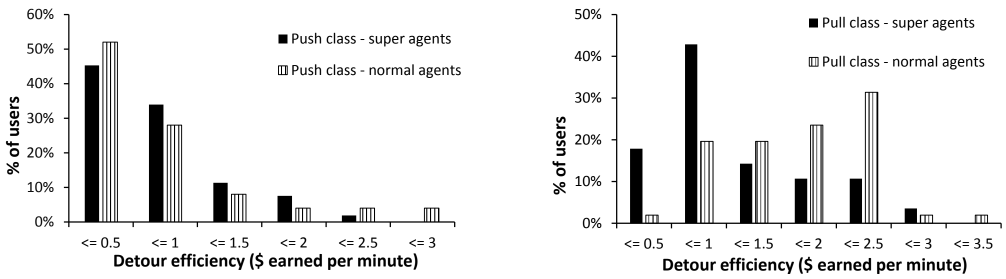


Fig. 7. Total detour incurred during the trial.

efficiency, which is defined as cents earned per detour minute. The histogram of detour efficiency for all active users is given in Figure 8. When broken down by the agent categories, super agents and normal agents have similar detour efficiencies at 169 and 170 cents per detour minute respectively. When broken down by the push/pull classes, the push class is earning 168 cents per detour minute, which is slightly more efficient than the pull class, which is earning 160 cents per detour minute. For normal agents in both push and pull classes, their performances are similar as well. However, the efficiency of super agents differs greatly depending on whether they are in the push or the pull class. While super agents in the push class earn 175 cents per detour minute, super agents in the pull class earn only 140 cents per detour minute, a 20% drop. This demonstrates that for dedicated mobile crowdworkers, push technology is particularly valuable as it allows workers to *outsource* their planning efforts to our recommendation engine.



(a) Push-class users.

(b) Pull-class users.

Fig. 8. Detour efficiency of (a) push-class and (b) pull-class users.

Task Selection Efficiency: By analyzing how users interact with our App, we can quantify the "search efforts" spent by users in identifying the set of tasks to commit to. More specifically, for each time window, we estimate the task selection efficiency within this time window as the time difference between the moment a user opens the App for the first time or browses through the list of tasks (whichever happens first) and the moment the user accepts a task. Similar to earlier metrics, we provide comparison across two dimensions: between push and pull classes, and between super and normal agents.

While users from the push class spent 9 minutes browsing through the tasks list and accept the tasks, users from the pull class spent nearly twice as long as their counterparts. This is consistent with our intuition that users from the push class have the advantage (over the pull class) of browsing only the tasks in the recommended list. Interestingly, despite the fact that super agents perform more tasks than normal agents, both agent classes spend 13 minutes (on average) in making initial task commitment. When analyzing both agent categories in each class separately, we notice that both super and normal agents spend much less time in the push class than in the pull class. For super/normal agents, it is 8/11 minutes for being in the push class and 15/18 minutes for being in the pull class.

Table 4. Summary of the metrics across classes and agent categories.

Metrics	Push vs Pull		Super vs Normal		Push		Pull	
	Push	Pull	Super	Normal	Super	Normal	Super	Normal
Total detour (min.)	5.2 (per task)	7.6 (per task)	45	15	40	12	62	18
Detour efficiency (cents / min.)	168	160	169	170	175	165	140	172
Task selection efficiency (min.)	9	16	13	13	8	11	15	18
Performance interval (min.)	8	3	6	6	8	9	3	3
Performance efficiency (distance)	one building	adjacent section	3 levels away	2 levels away	one building	3-4 levels away	2-3 sections away	adjacent section

Performance Interval: To understand how far in advance does a user commit to his tasks, we also measure the time in between task selection and task performance. We find that a user from push class submits a task after 8 minutes since the acceptance time; however, pull class users submit after a mere 3 minutes. However, agent category does not seem to play a role in this metric, as the performance intervals for both super and normal agents are around 6 minutes.

Performance Efficiency (distance): We also notice that users from the pull class accept tasks mostly in the vicinity of their current locations – when they are (on average) 50 seconds away from task locations (translates to several sections away, usually on the same building level). On the other hand, push class users are more likely to accept a task further away, even when they are several buildings away. Instead of reporting the actual travel time (or distance), we report whether the distance is such that it is still on the same level (but in sections), on different levels (but still in the same building), or in different buildings.

7 CONCLUSION

In this paper, we study a "push-based" paradigm for large-scale mobile crowdsourcing, where a centralized engine provides trajectory-aware task recommendations to a large pool of workers, while taking into account the inherent probabilistic uncertainty about their future trajectories. By using city-scale traces of commuter movement, we show that our Lagrangian-relaxation based heuristics can generate recommendations faster than an exact ILP formulation by more than two orders of magnitude, while suffering a less-than-1% degradation in the percentage of tasks recommended (compared to an idealized alternative where the realized routes are known a priori). Subsequently, we use real-world crowdsourcing studies on our campus to show that this push-based approach can be effective even with highly uncertain routine routes, achieving more than 25% of improvement in efficiency for top workers, compared to a traditional pull-based alternative.

Our results suggest that such centralized push-based task recommendations can be effective for real-world city-scale services, such as municipal monitoring and package delivery. In ongoing work, we are extending this model to consider the case of *task bundles*, where workers must pick an entire set of tasks. Such task bundling is becoming commonplace in urban delivery services as it helps workers amortize their travel overheads. An open question is whether such bundles can be effectively formed without knowing the route uncertainty of each worker, but based on the overall city-scale models of aggregate commuting flows.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative, and partially supported by the Xerox Research Centre India.

REFERENCES

[1] Florian Alt, Alireza Sahami Shirazi, Albrecht Schmidt, Urs Kramer, and Zahid Nawaz. 2010. Location-based crowdsourcing: extending crowdsourcing to the real world. In *6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*. 13–22.

[2] Claudia Archetti, Martin W. P. Savelsbergh, and M. Grazia Speranza. 2016. The Vehicle Routing Problem with Occasional Drivers. *European Journal of Operational Research* 254, 2 (2016), 472–480.

[3] Ioannis Boutsis and Vana Kalogeraki. 2014. On Task Assignment for Real-Time Reliable Crowdsourcing. In *34th IEEE International Conference on Distributed Computing Systems*.

[4] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. 1996. The team orienteering problem. *European Journal of Operational Research* 88, 3 (1996), 464–474.

[5] Cen Chen, Shih-Fen Cheng, Aldy Gunawan, Archan Misra, Koustuv Dasgupta, and Deepthi Chander. 2014. TRACCS: Trajectory-Aware Coordinated Urban Crowd-Sourcing. In *2nd AAAI Conference on Human Computation and Crowdsourcing*. 30–40.

[6] Marshall L Fisher. 1981. The Lagrangian relaxation method for solving integer programming problems. *Management Science* 27, 1 (1981), 1–18.

[7] Bruce L Golden, Larry Levy, and Rakesh Vohra. 1987. The orienteering problem. *Naval Research Logistics* 34, 3 (1987), 307–318.

[8] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. 2016. Orienteering Problem: A Survey of Recent Variants, Solution Approaches and Applications. *European Journal of Operational Research* 225, 2 (2016), 315–332.

[9] Jeff Howe. 2006. The rise of crowdsourcing. *Wired Magazine* 14, 6 (2006), 1–4.

[10] Shenggong Ji, Yu Zheng, and Tianrui Li. 2016. Urban sensing based on human mobility. In *2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 1040–1051.

[11] Yichuan Jiang. 2016. A survey of task allocation and load balancing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (2016), 585–599.

[12] Yichuan Jiang, Yifeng Zhou, and Yunpeng Li. 2015. Reliable task allocation with load balancing in multiplex networks. *ACM Transactions on Autonomous and Adaptive Systems* 10, 1 (2015), 3.

[13] Thivya Kandappu, Archan Misra, Shih-Fen Cheng, Nikita Jaiman, Randy Tandriansyah, Cen Chen, Hoong Chuin Lau, Deepthi Chander, and Koustuv Dasgupta. 2016. Campus-Scale Mobile Crowd-Tasking: Deployment & Behavioral Insights. In *19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. 800–812.

[14] Salil S. Kanhere. 2011. Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces. In *12th IEEE International Conference on Mobile Data Management*. 3–6.

[15] Leyla Kazemi and Cyrus Shahabi. 2011. A privacy-aware framework for participatory sensing. *ACM SIGKDD Explorations Newsletter* 13, 1 (2011), 43–51.

[16] Leyla Kazemi and Cyrus Shahabi. 2012. GeoCrowd: enabling query answering with spatial crowdsourcing. In *20th International Conference on Advances in Geographic Information Systems*. 189–198.

[17] Azeem J Khan, Vikash Ranjan, Trung-Tuan Luong, Rajesh Balan, and Archan Misra. 2013. Experiences with performance tradeoffs in practical, continuous indoor localization. In *14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 1–9.

[18] Artis Mednis, Girts Strazdins, Reinholds Zviedris, Georgijs Kanonirs, and Leo Selavo. 2011. Real-time Pothole Detection Using Android Smartphones with Accelerometers. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops*.

[19] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. 2008. Nericell: Rich Monitoring of Road Traffic Conditions Using Mobile Smartphones. In *7th ACM Conference on Embedded Networked Sensor Systems*.

[20] Mohamed Musthag and Deepak Ganesan. 2013. Labor dynamics in a mobile micro-task market. In *31st SIGCHI Conference on Human Factors in Computing Systems*. 641–650.

[21] Sarvapali D Ramchurn, Claudio Mezzetti, Andrea Giovannucci, Juan A Rodriguez-Aguilar, Rajdeep K Dash, and Nicholas R Jennings. 2009. Trust-based mechanisms for robust and efficient task allocation in the presence of execution uncertainty. *Journal of Artificial Intelligence Research* 35, 1 (2009), 119.

[22] Rajib Kumar Rana, Chun Tung Chou, Salil S Kanhere, Nirupama Bulusu, and Wen Hu. 2010. Ear-phone: an end-to-end participatory urban noise mapping system. In *9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. 105–116.

[23] John P. Rula, Vishnu Navda, Fabián E. Bustamante, Ranjita Bhagwan, and Saikat Guha. 2014. No "One-size Fits All": Towards a Principled Approach for Incentives in Mobile Crowdsourcing. In *15th Workshop on Mobile Computing Systems and Applications*. Article 3, 5 pages.

[24] Adam Sadilek, John Krumm, and Eric Horvitz. 2013. Crowdphysics: Planned and Opportunistic Crowdsourcing for Physical Tasks. In *7th International AAAI Conference on Weblogs and Social Media*. 536–545.

[25] Kate Starbird and Leysia Palen. 2011. Voluntweeters: Self-organizing by digital volunteers in times of crisis. In *29th SIGCHI Conference on Human Factors in Computing Systems*. 1071–1080.

[26] Matthias Stevens and Eliie D'Hondt. 2010. Crowdsourcing of Pollution Data Using Smartphones. In *Workshop on Ubiquitous Crowdsourcing, held at 12th ACM Conference on Ubiquitous Computing*.

[27] Jacob Thebault-Spieker, Loren G. Terveen, and Brent Hecht. 2015. Avoiding the South Side and the Suburbs: The Geography of Mobile Crowdsourcing Markets. In *18th ACM Conference on Computer Supported Cooperative Work & Social Computing*.

[28] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. 2009. Vtrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones. In *7th ACM Conference on Embedded Networked Sensor Systems*.

[29] Fabien Tricoire, Martin Romauch, Karl F Doerner, and Richard F Hartl. 2010. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research* 37, 2 (2010), 351–367.

[30] Khai N. Troung, Thariq Shihipar, and Daniel J. Wigdor. 2014. Slide to X: Unlocking the Potential of Smartphone Unlocking. In *32nd SIGCHI Conference on Human Factors in Computing Systems*.

[31] Theodore Tsiligirides. 1984. Heuristic Methods Applied to Orienteering. *The Journal of the Operational Research Society* 35, 9 (1984), 797–809.

[32] Rajan Vaish, Keith Wyngarden, Jingshu Chen, Brandon Cheung, and Michael S. Bernstein. 2014. Twitch Crowdsourcing: Crowd Contributions in Short Bursts of Time. In *32nd SIGCHI Conference on Human Factors in Computing Systems*.

[33] Jing Wang, Siamak Faridani, and Panagiotis Ipeirotis. 2011. Estimating the Completion Time of Crowdsourced Tasks Using Survival Analysis Models. In *Workshop on Crowdsourcing for Search and Data Mining*.

[34] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 25–34.

[35] Tingxin Yan, Matt Marzilli, Ryan Holmes, Deepak Ganesan, and Mark Corner. 2009. mCrowd: A Platform for Mobile Crowdsourcing. In *7th ACM Conference on Embedded Networked Sensor Systems*.

[36] Dejun Yang, Guoliang Xue, Xi Fang, and Jian Tang. 2012. Crowdsourcing to Smartphones: Incentive Mechanism Design for Mobile Phone Sensing. In *18th Annual International Conference on Mobile Computing and Networking*.

[37] Alexandros Zenonos, Sebastian Stein, and Nicholas R Jennings. 2015. Coordinating measurements for air pollution monitoring in participatory sensing settings. In *14th International Conference on Autonomous Agents and Multiagent Systems*. 493–501.

[38] Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology* 6, 3 (2015), 29.

[39] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology* 5, 3 (2014), 38.

[40] Yu Zheng, Tong Liu, Yilun Wang, Yanmin Zhu, Yanchi Liu, and Eric Chang. 2014. Diagnosing New York city's noises with ubiquitous data. In *2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 715–725.

Received December 2016; revised March 2017; accepted April 2017