Campus-Scale Mobile Crowd-Tasking: Deployment & Behavioral Insights

Thivya Kandappu†, Archan Misra†, Shih-Fen Cheng†, Nikita Jaiman†, Randy Tandriansiyah†, Cen Chen†, Hoong Chuin Lau†, Deepthi Chander*, Koustuv Dasgupta*

† School of Information Systems, Singapore Management University

∗ Xerox Research Centre, India

{thivyak@, archanm@, sfcheng@, nikitaj@, rtdaratan@, cenchen.2012@phdis., hclau@}smu.edu.sg,

{deepthi.chander, koustuv.dasgupta}@xerox.com

ABSTRACT

Mobile crowd-tasking markets are growing at an unprecedented rate with increasing number of smartphone users. Such platforms differ from their online counterparts in that they demand physical mobility and can benefit from smartphone processors and sensors for verification purposes. Despite the importance of such mobile crowd-tasking markets, little is known about the labor supply dynamics and mobility patterns of the users.

In this paper we design, develop and experiment with a realwprld mobile crowd-tasking platform, called *TA$Ker*. Our contributions are two-fold: (a) We develop *TA$Ker*, a system that allows us to empirically study the worker responses to push vs. pull strategies for task recommendation and selection. (b) We evaluate our system via experimentation with 80 real users on our campus, over a 4 week period with a corpus of over 1000 tasks. We then provide an in-depth analysis of labor supply, worker behavior & task selection preferences (including the phenomenon of super agents who complete large portions of the tasks) and the efficacy of push-based approaches that recommend tasks based on predicted movement patterns of individual workers.

INTRODUCTION

Mobile crowd-tasking, where a group of individuals utilize their smartphones to perform a variety of location-sensitive tasks, has become an increasingly popular computing paradigm. In particular, early evidence in favor of such crowd-tasking/sourcing[1] suggests that a variety of real-world, time-sensitive urban tasks can be performed more effectively, by utilizing a time-varying pool of "citizen volunteers", who opportunistically perform tasks that best match their own commuting and lifestyle patterns. Examples of such crowd-sourced tasks include retail sensing (e.g., reporting on the queuing delays at food courts and movie theaters), compliance and auditing (e.g., verifying if a product category is displayed in the right shelf in a convenience store) and logistics (e.g., picking up and dropping off packages within a business district).

Despite the compelling benefits of such a participatory and distributed model, it is difficult to guarantee a predictable level and quality of task execution, given the underlying uncertainties associated with crowd-worker participation. Research on mobile crowd-tasking thus addresses multiple aspects of this complex problem, including *task selection* (which tasks are best suited for which workers?), *task specification* (what types of location-specific tasks are users willing to perform?), *incentive design* (what pricing approaches effectively match available tasks to willing workers?) and *redundancy planning* (how does one quantify and mitigate the risks of workers failing to perform their agreed-upon tasks?) However, the overwhelming majority of such research is presently theoretical (typically evaluated by synthetic agent-based models), and fails to capture the real-world behavioral traits of an opportunistic pool of workers.

To gain more empirical understanding of this collaborative task execution paradigm, we have built and deployed a mobile crowd-tasking platform, called *TA$Ker*, on a large urban campus. *TA$Ker* provides voluntary student participants the opportunity to earn real monetary rewards by performing various *useful* campus-oriented tasks, such as "reporting on the cleanliness of restrooms", "verifying the availability of specific drinks in a vending machine" and "validating the crowdedness of study areas". While admittedly narrower than a city-wide deployment, the choice of the campus-scale setting is deliberate: it allows (a) longitudinal studies over a worker population that is persistent, and (b) fine-grained observations on user movement to be captured (via a Wi-Fi based location system) and incorporated into task selection strategies. While *TA$Ker's* back-end infrastructure is flexible enough to allow longer-term studies on different techniques for task selection, incentive design and redundancy planning, this paper focuses on the use of *TA$Ker* to experimentally study two different models of task selection:

- *Pull-based:* In this conventional decentralized model of crowd-sourcing (e.g., FieldAgent, GigWalk), individual users choose from the full set of available tasks (often selecting tasks that are close to the user's *current* location).

[1] In this paper, we use the terms "crowd-sourcing" and "crowd-tasking" interchangeably.

CSCW '16, February 27–March 02, 2016, San Francisco, CA, USA.

 ISBN 978-1-4503-3592-8/16/02…$15.00

DOI: http://dx.doi.org/10.1145/2818048.2819995

- *Push-based:* In this more modern, centrally-orchestrated model [2], the crowd-tasking engine recommends tasks that are based on an individual user's predicted movement trajectory (preferring tasks that minimize an individual's *detour* overhead). An individual then selects from this more-limited recommended set.

Given such an empirical crowd-tasking platform, there are several questions that we seek to empirically investigate:

- Are there any tangible overall benefits from the push-based (i.e., proactive recommendation) model, compared to the conventional pull-based approach, especially given the errors/uncertainties in location and movement estimation that are present in the real world? In particular, we hypothesize that the proposed model of recommending tasks so as to minimize expected detours may provide either higher rates of task acceptance/completion or lower travel overheads for the task-workers.

- Given the underlying variability in individual-level behaviors, how predictable is the overall task acceptance/completion pattern (across days, at different locations)? And how does this performance metric depend on other factors, such as the type of task, the demographic attributes of task-workers, just to name a few?

- Are there observable behavioral differences between the push-based vs. the pull-based models, in the way task-workers accept and perform tasks? And, can such differences lead to better models for quantifying the risk of task dereliction; i.e., the likelihood that users will not complete assigned tasks within the stipulated time frame? In particular, we hypothesize that push-based recommendations may allow workers to accept tasks with higher lead time (especially if they anticipate that their itinerary will take them to the task location at a later time) vs. pull-based models where workers are likely to be more myopic and opportunistic (selecting tasks that are near their current location and thus they can complete "right away").

Key Contributions: This paper reports on both the design and empirical experiences with the deployment of the *TA$Ker* App, to a pool of approximately 80 undergraduate students over a period of around 4 weeks. We make the following major contributions:

- *Feasibility of Campus-based Mobile Crowd-tasking:* We show that our designed and deployed *TA$Ker* App actually works: 80 users used it, over the initial launch period of 4 weeks, to perform a total of 800 tasks, and earn real rewards of $800. We shall show how appropriately designed route-prediction and task-recommendation engines allow us to perform reasonably accurate detour-minimizing recommendations, even under (a) location errors of $\pm 6-8$ meters that are typical of real-world location technologies, and (b) real-world uncertainty in the on-campus movement of the users.

- *Superiority of Trajectory-aware Proactive Recommendations:* We empirically demonstrate that the performance of urban crowd-tasking is significantly better (via statistical measures) when tasks are proactively recommended vs. when individuals pull tasks without any central coordination. In particular, we show that the push-based model achieves better task completion rates (56% vs. 44%), and also results in a lower average detour overhead (3.8 mins vs. 5.4 mins).

- *Certain tasks are preferred over others:* We discover that the ease of executing tasks (using the smartphone) plays an importance role in the overall acceptance and completion rate of tasks–e.g., tasks that involve simple multiple-choice based questions achieve 40 times higher completion rates, compared to tasks that require the use of the camera. Note that the differences in task execution time (about 60 secs) are much smaller than the travel detour overheads (approximately 5 minutes). However, users prefer to perform tasks whose actual execution is simpler, even if the task itself might require them to take a disproportionately longer detour.

- *Planned vs. Impulsive Task Performance:* We observe a clear difference in the task acceptance and execution dynamics between push-based vs. pull-based workers. In particular, push-based workers often accept tasks in advance, perhaps anticipating that they will eventually be at the relevant task location. In contrast, pull-based workers exhibit a significantly more opportunistic execution pattern: on average, they accept a task only 3 minutes before its actual execution and when they are only 40 seconds away (i.e., when they are located in a section that is adjacent to the task's location) from the task location.

- *Behavioral Differences between Regular and "Super-Agents":* Similar to results reported by Musthag and Ganesan [11], we observe the existence of *super agents*, a small set of workers who perform a significant majority of tasks. We empirically show that the higher rewards of super agents come because they are willing to undertake significantly longer total detours (approx. 10 times higher) compared to regular workers. Moreover, we additionally demonstrate that push-based super agents are more efficient (earn 25% more per unit detour overhead) than pull-based super agents.

We like to emphasize our twin goals for this paper: (a) to gain an empirical understanding of the technological feasibility of such campus-level crowd-tasking, and (b) to discover key behavioral patterns related to mobile crowd-tasking, and thus help shape our community's research agenda around this collaborative computing model.

RELATED WORK

Mobile crowd-sourcing is a relatively recent phenomenon that allows workers to contribute by physically moving to specified locations and performing the relevant tasks. There are already a number of well-established commercial operators, such as FieldAgent [2], GigWalk [3], and NeighborFavor [4].

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

In most of these commercial operations, workers *pull* tasks by browsing through task listings sorted by proximity to their current locations.

Task Assignment: Some researchers have proposed to improve current practices by making the process more interactive; for example, Kazemi et al. [6] propose an iterative assignment framework where central operators query workers near a task one by one in order to determine worker's availability. All pull-based approaches suffer from what Musthag and Ganesan [11] describe as the *super-agent phenomenon*; i.e., a small percentage (less than 10%) of workers dominating the task selection, and thereby leading to progressive dropout by the rest of the worker population.

Follow-up studies by Kokkalis et al. [9] and Talamadupula et al. [12] showed that one potential solution to counter such undesirable outcomes is to provide workers with recommended action plans. A recently proposed framework, called TRACCS (including both the deterministic version [2] and the more recent stochastic variant [3]), builds on top of these insights, and utilizes knowledge about a worker's usual movement patterns to produce task recommendations. Broadly speaking, these approaches seek to improve the decentralized or first-come-first-served models of *task assignment*, a topic that Kittur et al. [8] have identified as critical to the sustainable operation of future crowd-tasking platforms. However, such academic research is quite disconnected from the pull-based approaches that current mobile crowd-tasking platforms employ.

Task Acceptance & Completion Behavior: Limited studies have explored the relationship between task attributes and worker behavior, especially for the crowd-sourced execution of location-dependent tasks. Wang [13] 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 [5] 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 crowd-tasking 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.

SYSTEM DESIGN

Our long-term goal is to make our *TA$Ker* App an experimental platform to empirically investigate a variety of open issues related to mobile crowd-tasking, with emphasis on (A) better task recommendation strategies, and (B) better design of incentives to facilitate higher task execution rates. While the overall *TA$Ker* architecture is flexible enough to permit experimentation on both dimensions, in this paper, we focus principally on goal (A), i.e., on studying alternative task recommendation/ selection strategies (specifically push vs. pull models) and the corresponding behavior of crowd-workers.

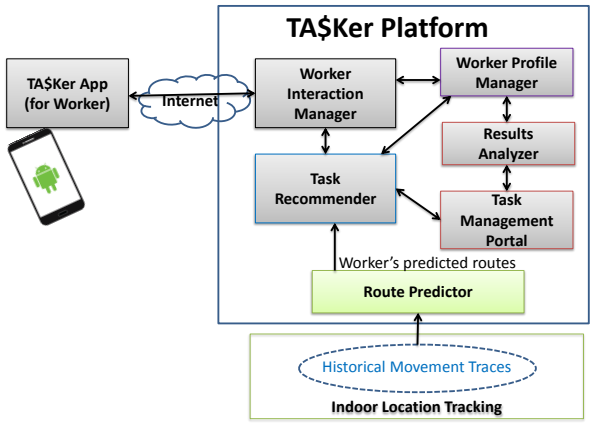


Figure 1. Overall architecture of the *TA$Ker* framework.

At the outset we recognize that *TA$Ker's* campus-based crowd-tasking model (deployed over 5-6 university buildings and a maximum population of 2000-3000 students) is significantly smaller, both in terms of spatial scale and number of participants, than a true city-scale mobile crowd-tasking service. However, we believe that *TA$Ker* provides an invaluable platform for experimental studies, as our campus environment and student participants allow for a far higher level of rigorous and longitudinal experimentation than would be otherwise possible.

Fig. 1 shows the overall architecture of the proposed framework, and illustrates the various individual components and their interactions. To support the push-based recommendation models (such as the ones proposed in [2] and [3]), the server infrastructure needs to (a) perform longitudinal monitoring of an individual's location traces, and then predict the individual's likely movement trajectories, and (b) implement a centralized model for recommending tasks to each individual, based on these predicted movement trajectories. The key functional components of the *TA$Ker* infrastructure include:

- *Task Management Portal*: This back-end server component allows system administrators or task owners to specify various attributes of tasks, such as the time window or completion deadline by when a task must be performed), the associated rewards, the task's location and the *redundancy factor* (the number of independent workers per task required to achieve the required levels of "truth discovery").
- *Route Predictor*: This utilizes historical traces of individual user movement to develop a predictive trajectory profile. Note that this component is needed only for the proac-

tive push-based model of task recommendation. To provide such predictions, this component assumes the existence of an underlying *Location Tracking Infrastructure* (outlined separately in Fig. 1). More specifically, *TA$Ker* takes advantage of a campus-wide, server-side Wi-Fi based indoor location tracking system (similar to approaches described in [10]) that has been operationally deployed for the past 18 months, which provides continuous location tracking of all Wi-Fi connected mobile devices (with a median location error of $\pm 6-7$ meters and a refresh rate of approximately 2 minutes).

- *Task Recommender*: This component is responsible for taking as inputs both (a) the list of location-dependent tasks, and (b) the predicted movement profiles of workers, and then determining the set of tasks recommended to each individual worker. Each crowd-worker may, of course, choose to accept (or not accept) one or more of these recommended tasks. For the pull-based model of operation, the Recommender's job is to simply list the entire set of available tasks (irrespective of the task's on-campus location or its time-window constraints). For the push-based model, however, the Recommender must implement an algorithm that provides each worker a smaller set of recommended tasks, that are best suited to each individual's *predicted* movement pattern. This component is implemented modularly, with different recommendation algorithms implemented as add-on libraries.

- *Worker Interaction Manager*: This server component is responsible for handling interactions of individual workers with back-end components via the *TA$Ker* mobile App. Such interactions include providing the App with a list of recommended tasks, capturing the results of the executed tasks (i.e., the values or content reported by the crowd-worker) and capturing the details of the individual crowd-worker's interaction with the mobile App (e.g., the time taken to view tasks before selecting them).

- *Results Analyzer:* This server component is used to analyze the results of various experimental strategies, and thus deduce insights into the effectiveness of various task recommendation/incentivization strategies and their effects on individual/aggregated crowd-worker/s behavior. This component applies appropriate statistical analysis over the datasets captured by various other server-side components.

- *The* TA$Ker *App:* Finally, this is a client-facing mobile App that crowd-workers use to view available/recommended tasks, select tasks to perform, execute those tasks (e.g., enter free-form text or upload pictures), and also view various outcomes (such as the amount of rewards earned). This App interacts with the back-end Worker Interaction Manager component.

Our experimental investigation of different recommendation and reward pricing strategies requires us to iteratively modify the policies/algorithms in the *Task Management* and *Task Recommender* components. For example, to investigate a different recommendation strategy for a subset of workers, we simply need to specify the use of a different add-on library for that pool of workers. While this overall architecture is conceptually similar to other crowd-tasking frameworks, we would like to emphasize that: (a) By explicitly incorporating real-world movement profiles and permitting easy selection of different recommendation algorithms, our architecture is among the first that is geared to experimentally study the impact of different push-based recommendation strategies; and (b) Developing a real-world platform that supports useful push-based recommendation is not a trivial goal, as the components must be designed to handle the real-world location errors and the "seemingly random" movement behaviors of individuals in campus environments. In fact, in the next section, we focus especially on the *Route Predictor* and *Task Recommender* components and discuss how they are designed to remain useful even under real-world noise, location errors, and uncertainties of movement behavior.

ROUTE PREDICTION ENGINE

In this section we develop a route prediction algorithm that utilizes historical mobility traces of users to generate routes that a user will choose within a time window. We then explain how we gathered indoor mobility traces, extracted *reference* locations and found regular routes.

The Mobility Traces Data

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 [7]), and provides a median accuracy of 6-8 meters; due to limitations on the commercial Wi-Fi infrastructure, the location updates occur sporadically, with a mean interval of approximately 3 minutes.

We consider the location data from a full academic term from January to April in 2015. This movement data was captured for *all* Wi-Fi connected users on campus and consists of an average of 9,000 distinct devices observed daily. From this raw data we extracted reference locations and routes, a process we describe next.

From Traces to Routes

Our route prediction algorithm depends on knowledge of a user's past trips, but the data gives no explicit indication of when a trip begins or ends. This section explains how we go from raw data to a plausible set of routes per user.

For the purposes of this paper, we define a *route* to be a sequence of locations on campus in the order of visits. We create a three-stage process to transform the raw data to routes: first, we extract reference locations in which user stays more in a time segment (30 minutes in our case); second, we formulate the movement of a user for a given time window (e.g., 9am – 12pm) as a transition graph; and third we find the best

route represented as a sequence of reference locations that maximize the likelihood that this route would generate all observations we have seen from the data of a user. The entire route is then generated by finding the shortest path between the two identified reference locations of adjacent time segments. These steps are described in detail below.

Reference Location Extraction: 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.

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, 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's possible that $i = j$, which indicates that a user stays 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 $(t+1)$.

Route Prediction: Using the transition graph, the likelihood of each route can be simply calculated by multiplying all probabilities of edges 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 75% of users.

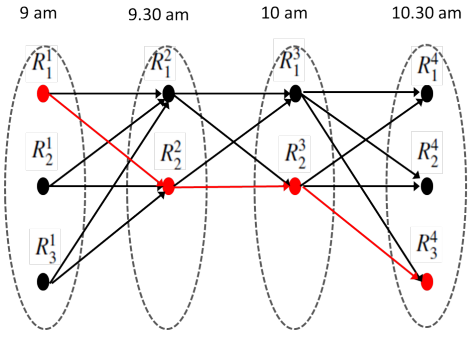


Figure 2. A sample transition graph.

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 Fig. 2), 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 line in Fig. 2.

Prediction Accuracy

The accuracy of our prediction is evaluated based on the mobility traces of 80 students who installed our App (described later in §Prototype Implementation). We split our data into two parts for training and testing. The training data comes from the month of February, and the testing data comes from the first week of March. For each weekday (Monday to Friday), a transition graph is constructed for each user based on four weekdays in February. The generated predictions for each weekday in the first week of March is then compared against the ground truth. We observe that our prediction algorithm achieves 68% precision and 75% recall.

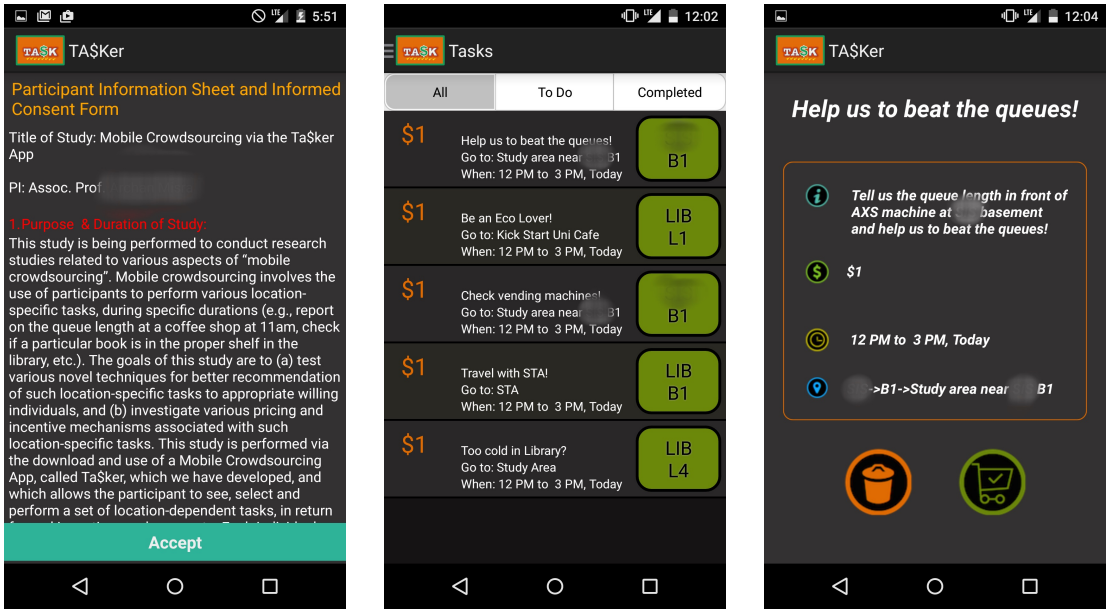
RECOMMENDATION ENGINE

In this section, we present the algorithm that is implemented in the recommendation engine. As reviewed earlier, most existing commercial mobile crowd-tasking platforms typically employ a "pull-based" model, where workers have to manually browse through and select from a list of available tasks. As demonstrated by recent research, such as the model proposed by Chen et al. [3], the efficiency and effectiveness of a mobile crowd-tasking platform can be significantly improved if task recommendations are "pushed" to workers based on predictions of worker's trajectory.

The input to the algorithm is a tuple comprising three elements: (1) a graph representing the geographical network; (2) worker information (each worker's desired detour time and the probability distribution of predicted routine routes); and (3) task information (task location, estimated task execution time, and reward). The output is a list of task recommendations for each worker.

More specifically, we follow the formulation from [3], and formulate the above task recommendation problem as a stochastic integer linear programming (ILP) model, whose objective is to maximize the expected rewards collected by all workers under route uncertainties. This ILP formulation can be seen as a specialized routing problem which recommends tasks to workers subject to a set of worker-specific time budget constraints, route connectivity constraints, and predicted route requirements.

The ILP model can be solved by using standard solvers such as CPLEX up to limited problem size. To scale up the solution approach to real world problem, we apply a well-known Lagrangian Relaxation (LR) heuristics [4] on a set of global constraints connecting the task recommendation and task completion decision variables (adopted from [3]). Having observed a clear worker-route level separable structure in the formulation, we further decompose the relaxed problem into two classes of much smaller subproblem: a recommendation



(a) Consent form. (b) All tasks. (c) Detailed task description.

Figure 3. Screenshots of Android App showing: (a) the consent form to be accepted, (b) list of available tasks, and (c) perform screen.

subproblem and worker-route level routing subproblem for each worker's predicted routine route.

The performance of this LR-based method depends on how subproblems are solved. Following [3], we solve each subproblem by a greedy heuristic to further improve scalability. Each worker starts with one of its routine routes, and the heuristic repeatedly inserts a task into the routine route at a position that maximizes the gain among all feasible insertions. The LR-based method progresses iteratively using a standard subgradient descent algorithm. This method is denoted as *LR-Greedy*. To further speed up this method, some *performance-boosting preprocessing* steps are implemented: a) for each subproblem, infeasible tasks are excluded (based on the amount of detour time), and b) all-pair shortest paths are pre-computed.

By default, a task is recommended to at most one worker. To enhance the robustness of the recommendation scheme (in case a worker fails to carry out some committed tasks, or the quality of the work is low), we might want to recommend a task to more than one worker. This requires only minor modification to the model, and is controlled by a parameter η, which means that we want to ensure that a task is recommended to at least η agents. This newly added constraint will be all-or-nothing, i.e., we will either recommend η agents to a task, or not at all if this is not possible. The objective is also slightly modified to maximize the expected reward collected by all the workers, but now normalized to η (since each completed task will be executed η times).

To make the recommendation algorithm work on our testbed campus, we divide the whole campus into 165 smaller sections/locations and physically map all locations into a graph. Each node in the graph represents a unique location and each weighted edge represents the travel time between two directly connected nodes (measured in minutes).

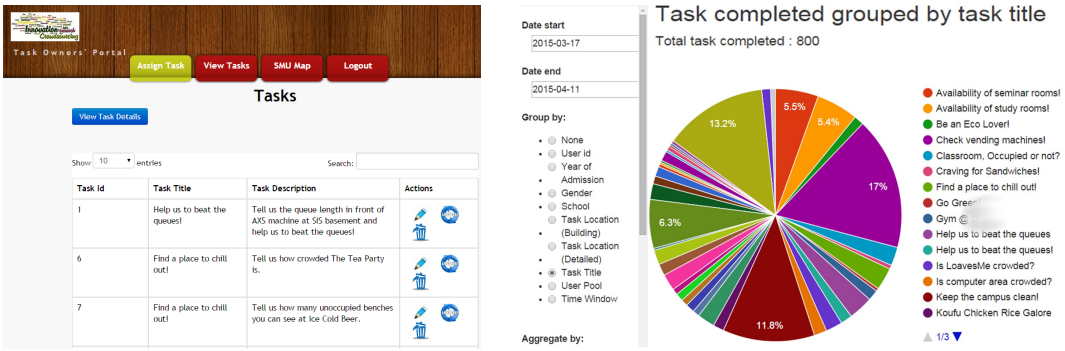
In this trial, we assume that execution time for each task is 5 minutes, and each worker has a detour time budget of 10 minutes over a three-hour planning horizon.

PROTOTYPE IMPLEMENTATION

We now describe the prototype we built to evaluate our system with real users. We then discuss interesting insights obtained from a four-week trial involving 80 campus students (whose historical location traces are available to us).

System Components

Our *TA$Ker* prototype consists of two parts, conforming to the architecture presented earlier in Fig. 1 – a front-end Android application for users to perform tasks, and a back-end database/server that stores user data and communicates with the App using HTTP messages. Screenshots for the Android App (originally built on Android 3.0 and above) are shown in Fig. 3. The opening screen allows the user to read and accept the consent form (mandatory to proceed with the App) (Fig. 3(a)) followed by the terms and conditions of the App. A list of the available tasks is shown on the next screen (Fig. 3(b)), along with the details of task location and the validity time period – in this example the user clicks the *Help us to beat the queues!* task and the relevant task description pops up. Tapping *Accept* will add the task in the user's *To Do* list to be performed during the task's validity period. The perform screen (Fig. 3(c)) lists the detailed description of the task, giving user the flexibility to either perform and submit the relevant response to the server or delete the task from the *To Do* list. The *About-Us* screen provides an overview about the *TA$Ker* App.



(a) List of tasks. (b) Graphical results.

Figure 4. Screenshots of task owner portal showing: (a) the list of tasks posted by the owner, and (b) graphical results.

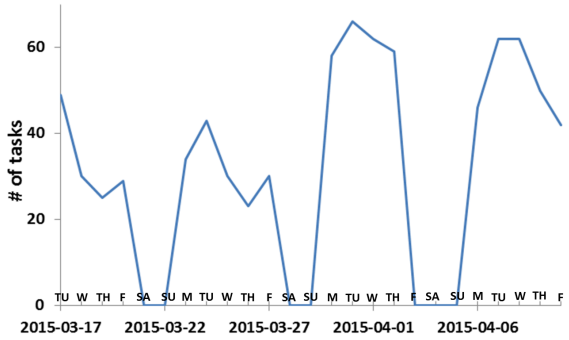


Figure 5. Number of completed tasks in a day.

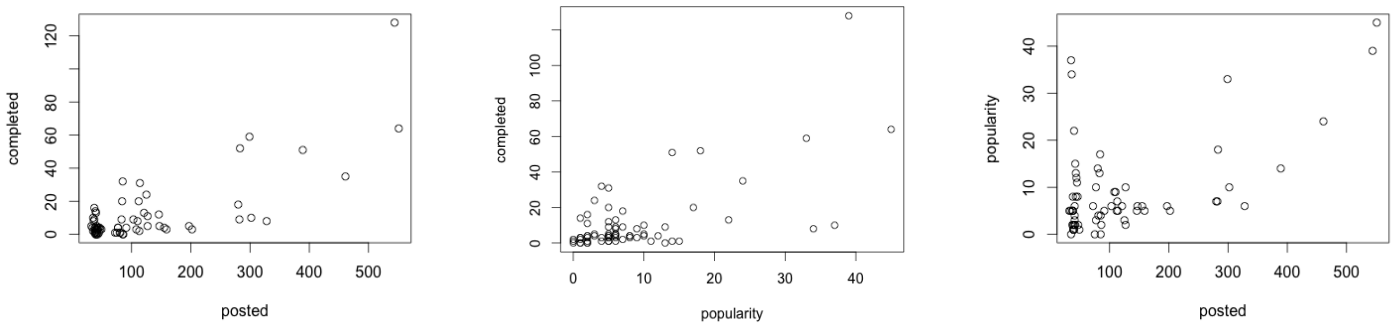
The server was implemented using the CodeIgniter web framework (written in PHP) and uses a MySQL database to store various data related to users, tasks and the user interactions with the tasks. Tasks (and associated details) are entered into the database via an Web interface accessible only to administrators. Fig. 4 shows screen shots of the web interface. The first screen (Fig. 4(a)) shows the list of created tasks. This same interface allows individual task-workers to be assigned to different control/treatment classes for field experiments. Once a task is completed (i.e., when the task receives the expected number of responses), the *Results Analyzer* component computes graphical results (shown in Fig. 4(b)) that can be studied along multiple dimensions, such as user demographics, task location and task validity duration.

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 80 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 four-week period towards the end of the academic session. 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, 80 students were randomly and equally divided into two pools "Push" and "Pull". For students belonging to the "push" class, they were provided task recommendations tailored to their respected itineraries (identified from historical location traces). 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 provide users a diversity of tasks with execution deadlines broadly aligned to student's typical schedule on campus, each day was divided into three 3-hour time windows: (a) 9am – 12pm, (b) 12pm – 3pm, and (c) 3pm – 6pm. New tasks were pulled 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 (location, time and task type independent) reward structure: every successfully completed task resulted in an earning of $1.
- *Max. Tasks per Time Window:* The *TA$Ker* back-end server was configured to allow an individual the ability to perform at most $Max(t)$ tasks within a time window t. This upper bound was deliberately specified to prevent scenarios where students began to neglect their academic duties and focus instead on obtaining larger rewards by performing a large number of tasks. Each student was allowed to perform at most 2 and 3 tasks per time window during the first and last two-weeks of our trial period, respectively.
- *Varying* η*:* During the 4-week study period, η (the minimum set of workers needed to be assigned per task) was varied, starting from $\eta = 1$ and increasing by 1 each week. This allowed us to examine the effect of η on the task completion rate.
- *Types of Tasks:* To maintain a constantly evolving, diverse set of tasks, and to also study the response of workers to different task types, our studies involve 4 distinct task categories:



(a) Posted tasks vs. completed tasks. (b) Popularity vs. completed tasks. (c) Posted tasks vs. popularity.

Figure 6. Demographic differences of super agents.

1 *Discrete-valued multiple choice tasks*: Here, the user selects the most appropriate answer (usually boolean-valued) from a predefined set of categorical values (e.g., reporting on the availability of group study rooms, checking the availability of snacks in a vending machine).
2 *Counting based tasks*: Here, the user had to enter a numerical value (e.g., report the queue length in front of a food stall, or count the unoccupied benches in a study area).
3 *Free-text based tasks*: Here, the user had to key-in some free-form textual description (e.g., report on the cleanliness of the restrooms, check the price of pastries in the coffee shop).
4 *Picture taking tasks*: Here, the user was asked to visit the task location and upload a picture taken with his smartphone camera (e.g., taking a photo of the corridor of a building, click a picture outside the cafe).

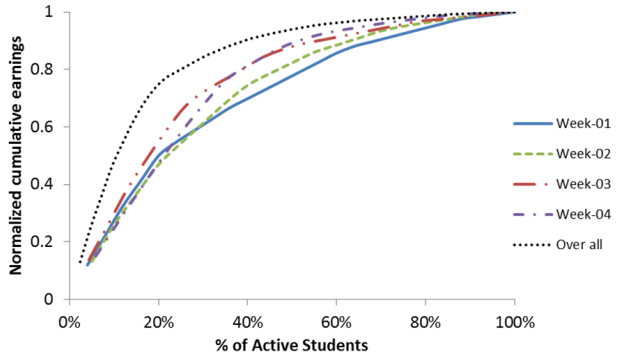


Figure 7. Super-agent phenomenon.

RESULTS AND KEY INSIGHTS

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

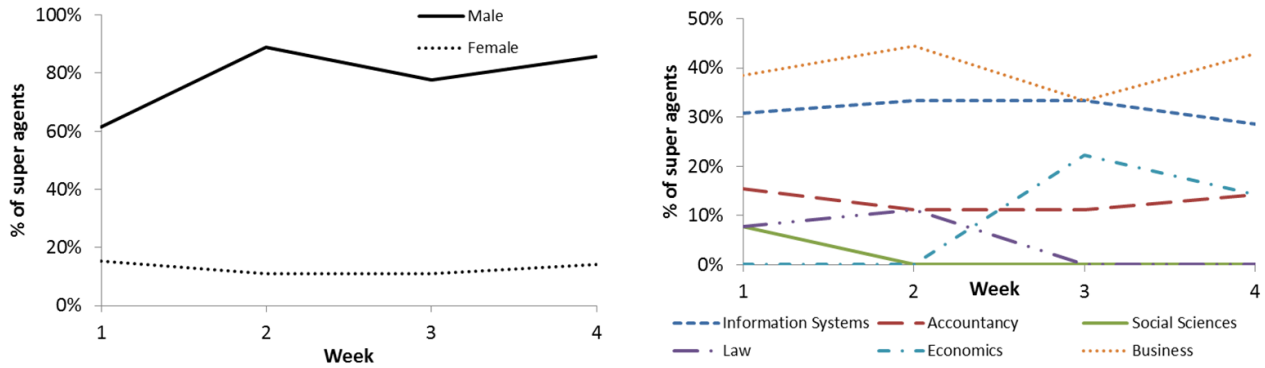
Demographics

During the four-week trial period, more than 1000 tasks were posted and 800 were completed by fifty active users (refer to Fig. 5 to see the number of completed tasks on a daily basis). Active users are those who have completed at least one task. Each task can only be performed by at most one user and a user can perform at most nine tasks in a day (three tasks per time window). No tasks were posted during the weekends and Easter Friday (4th of April, 2015).

We find that there are significantly more male active users (75%) than their female counterparts. Similarly, we could also notice that most active users come from the School of Business of this university (31% – normalized over the number of registered students from the same school), followed by the School of Information Systems.

We next investigate how the properties of a location or the number of posted tasks affected the corresponding number of completed tasks. Fig. 6 shows the scatter plots (pairwise) among the three variables: (i) number of posted tasks, (ii) number of completed tasks, and (iii) a location's *popularity* (measured by average number of active users at that location). Clearly, more popular places had a larger number of tasks posted (and completed). To further understand this phenomenon, we computed the correlation, jointly, between the number of completed tasks in each location (as the dependent variable) and (i) the number of posted tasks and (ii) the location's popularity. The corresponding correlation coefficients were $\rho = 0.096$ (for number of posted tasks) and $\rho = 0.67$ (for location popularity), indicating that the likelihood of task completion is significantly more affected by the overall occupancy levels of the task's location, rather than the volume of available tasks.

Push-class students are more active in completing tasks: As expected, we find more tasks are completed by users from push class (56% of the total completed tasks). This is despite the fact that this class has fewer active students than pull class (22 in push vs. 28 in pull). A t-test confirms that push class completes significantly more tasks than pull class, with p-value less than 0.0001.



(a) Demographics: Gender.

(b) Demographics: Field of study.

Figure 8. Demographic differences of super agents.

The tasks are mainly performed in library: Our campus library building includes many prime locations where many student activities are usually carried out (and thus exhibit highest occupancy counts), which explains why most number of the tasks were completed (43%) in this building as anticipated.

Lunch breaks are good times to perform tasks: The largest plurality (37%) of completed tasks occurred during the second time window (12pm – 3pm). Considering the fact that the first and last time windows are the peak-hours for lectures, it's not surprising that the second time window accounts for more tasks completed. This result is consistent with the findings in [1], which reported that crowd-workers preferred to perform tasks outside their core business hours.

Picture-based tasks are least popular: In Table 1, we tabulate the task completion statistics categorized based on task type, such as, percentage of completed tasks, detour incurred and execution time. Interestingly, multiple-choice question type was the most popular task category among the participants (40% of the completed tasks belong to this category – as a proportion of total tasks), while picture-based tasks are least popular (only 1.6% of completed task belong to this category – as a proportion of total tasks). One explanation for this might be the task execution time: the average task execution time for picture-taking tasks is three times longer compared to multiple-choice tasks.

Table 1. Task completion statistics.

Task type	% of comp. tasks	Detour (in min)	Execution time (in secs)
Multiple-choice	40.31%	5.7	30
Counting	38.6%	2.4	48
Text	19.48%	2.5	72
Photo	1.6%	0.5	96

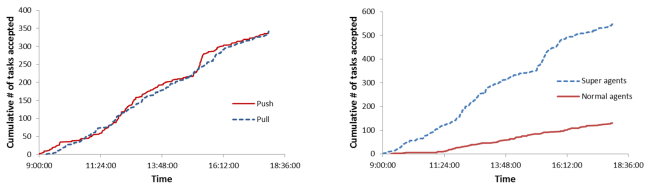
Another interesting finding was that users prefer to perform tasks whose actual execution is simpler, even if the task itself might require them to take a disproportionally longer detour. For example, although picture-taking tasks take much longer to complete, the required detour time is actually much shorter than multiple-choice tasks (30 seconds for picture-taking tasks vs. 5.7 minutes for multiple-choice tasks). This might be due to the fact that 75% of the completed photo tasks were performed by push class users.

Super-Agent Phenomenon

While examining *TA$Ker* users' behavior, we observe an interesting trend – a relatively small core-group of users generate a disproportionally large fraction of task responses. Fig. 7 shows that 25% of active agents are responsible for 80% of total earnings or total tasks done on *TA$Ker* (depicted by dotted line). 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 25% of active agents as *super agents*.

We investigate how the size of the group of super agents varies across weeks (depicted by four color lines in Fig. 7). We observe that as the weeks progress and more students register for the App, the percentage of super agents gradually decreases. However, with an increasing pool of available tasks, the average number of tasks completed *per super agent* increases as the weeks passby.

Further, we examine the demographic (i.e., gender and field of study) differences among super agents on a weekly basis. It is interesting to see that in Fig. 8(a) and Fig. 8(b), the majority of the super agents belong to male category (75%) and to the School of Business on our campus (42%).



(a) Push vs. pull.

(b) Super vs. normal users.

Figure 9. Task acceptance time.

To study the task accepting and submitting nature (e.g., browsing time, acceptance time, types of tasks performed) of individuals, in Fig. 9, we show how the cumulative number of accepted tasks evolves with time. From Fig. 9(a) we

Table 2. Summary of the metrics across classes and agent categories.

Metrics	Push vs Pull		Super vs Normal		Push Class		Pull Class	
	Push	Pull	Super	Normal	Super	Normal	Super	Normal
Total detour (in min.)	3.8 (avg. per task)	5.4 (avg. per task)	175	18	131	21	210	15
Detour efficiency (in cents per min.)	28	25	27	26	27	28	21	28
Task selection efficiency (in min.)	10	20	15	15	10	11	22	19
Performance interval (in min.)	9	3	6	5	10	9	4	3
Performance efficiency (in distance)	one building	adjacent section	3 levels away	2 levels away	one building	3-4 levels away	2-3 sections away	adjacent section

see that the difference between push and pull class users in accepting tasks along time is not very significant. However, as anticipated, in Fig. 9(b), super agents outperform normal agents by accepting tasks at a higher rate as time evolves.

Efficiency of Users

In this section, we define and study user's efficiency in performing mobile crowd-tasking tasks. If we know exactly how long a user spent in detours when performing selected tasks, we can straightforwardly define his *detour efficiency* as dollars earned per minute of detour. To further understand the user's efficiency during different stage of task performance, we can further split his time spent into two stages: planning, which refers to his efforts involved in browsing and selecting tasks on the App, and physical movement, which refers to his actual physical efforts in moving to and performing selected tasks. Table 2 summarizes the metrics we considered to compare the users across push and pull classes, agent categories and users' behavior (super agents and non super agents behavior when they are assigned to push and pull mode).

Detour: To compute detour efficiency, we first 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 first need to identify the neighboring stay locations (both prior to and after the task performance) in which he 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.

Fig. 10 shows the histogram of the total detour made per user throughout our trial period. We find that users from the push class incurred a detour of 3.8 minutes per task, which is 1.6 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

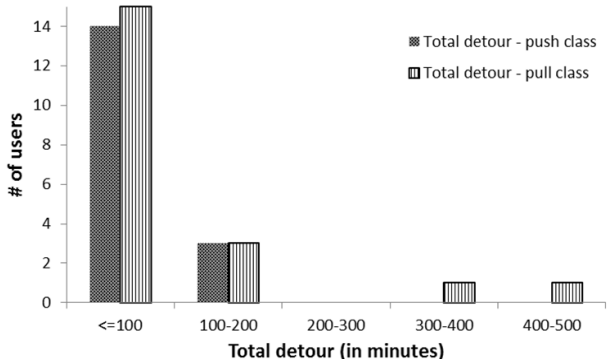


Figure 10. Total detour incurred during the trial.

average 175 minutes while an ordinary agent contributed only 18 minutes.

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 130 minutes of detour while normal agents incurred only 20 minutes. The same trend is observed in the pull class – super agents made significantly more detour compared to normal users (210 and 15 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? Fig. 11 shows the histogram of detour efficiency for all active users (i.e., who have at least completed one task during our trial). We find that super agents have higher detour efficiency in all except the first two quartiles of agents, where the trend flips. This demonstrates that that super agents are actually willing to take longer detours (and thus sacrifice detour efficiency), so as to complete a larger number of tasks. The higher detour efficiency for some other users also arises from the fact that they perform only a few tasks that are indeed very close to their stay locations. In terms of average detour efficiency (measured by cents earned per unit detour), super agents and others earn on average 27 and 26 cents per minute, respectively.

We find that push and pull class users earn 28 and 25 cents in a minute, respectively. However, super agents in the pull class

have significantly lower (about 25% lower detour efficiency) than regular agents in the pull class, while this discrepancy is noticeably absent in the push class. Thus, a benefit of push-based crowd-tasking is that *it allows super agents to select and perform a larger number of tasks without incurring a disproportionally higher detour overhead.* The amount of money earned per unit detour by agents in both categories (super and normal) is same regardless of the classes (push and pull) they belong to.

Temporal Efficiency: Having looked at the mobility patterns of users in *TA$Ker* , we now turn to the temporal aspects of their behaviors. Specifically, we look at how efficiently they use their time on *TA$Ker* . The time that a user spends on the *TA$Ker* App can be split into two parts: (a) the time that a user spends on going through the list of available tasks to search for tasks to commit to, and (b) the time that a user spends on performing a task, regardless of its type. We are able to accurately compute time allocation in these two parts since our App collects all activity traces when users interact with our App.

Task Selection Efficiency: The temporal efficiency of a user not only depends on the time spent in performing the tasks, but also on the time he spent looking for the tasks to perform. In this metric we look at the amount of time a user spends on searching and planning. We define the task selection efficiency as the amount of time it takes to find an appropriate task from the list of available tasks and accept it. For each time window, we estimate time selection efficiency within it 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 he accepts a task. Similar to earlier metrics, we provide comparison across two dimensions: between push and pull classes, and between super agents and ordinary users.

While users from push class spent 10 minutes browsing through the tasks list and accept the tasks, users from pull class spent twice as long as their counterparts. This is consistent with our intuition that users from push class have the advantage (over pull class) of browsing only the tasks in the recommended list. However, similar trend can not be observed between super agents and ordinary users. Both super agents and ordinary users spend 15 minutes in task planning. This is partly due to the fact that super agents come from both push and pull classes, and partly due to the earlier confirmed fact that super agents are willing to spend more time.

While analyzing both agent categories in each class separately, we notice that super agents in push class outsmart the super agents in pull class while taking lesser time to select the relevant tasks to perform (10 minutes for push class super agents, which is two times lower than the super agents in pull class). The similar trend is observed between the normal agents in push and pull classes (11 and 19 minutes respectively). This proves our intuition again that regardless of the super agent behavior, users in push class are benefited by browsing the tasks only in the recommended list.

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 the task performance. We find that a user from push class submits a task after 9 minutes since the acceptance time; however, pull class users submit after a mere 3 minutes. However, a similar trend is not observed between super agents and normal users. Both categories of users perform the tasks after 5 minutes since the acceptance time. Super agents in the pull class submitted tasks quicker (in 4 minutes) than super agents in the push class (in 10 minutes). The normal agents also behaved similarly, confirming that pull class users are being more opportunistic while performing tasks.

We also notice that users from the pull class mostly search for jobs in the vicinity of their current location – when they are only 40 seconds (on average) away from task locations, and accept tasks to perform immediately. On the other hand, push-class users are more likely to accept a task further away, even when they are several buildings away.

Task Failure Analysis

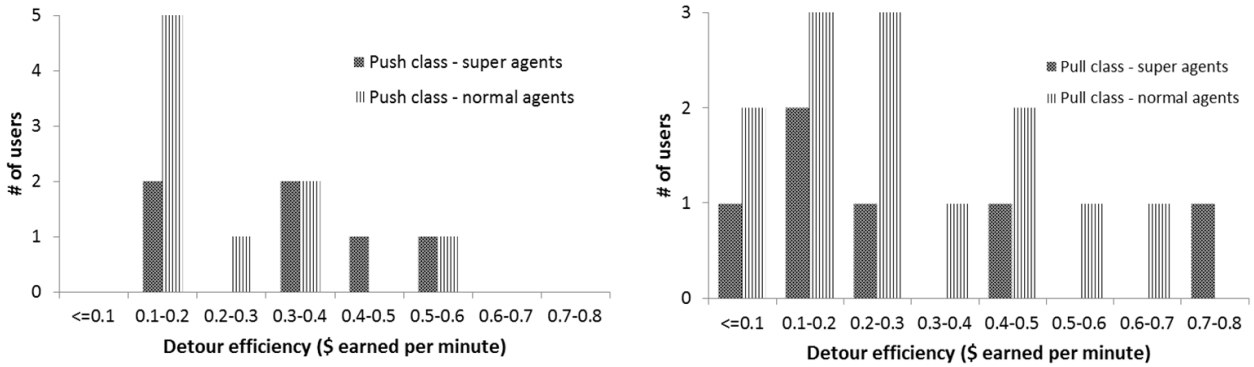
Finally, we investigate why some tasks are accepted yet not completed. Overall we observe that 49 tasks were accepted but not performed by users. Based on the tracking history, we observe that 34 tasks were accepted but responsible users did not even open the App until the time window had expired, despite the reminder notifications that were sent 15 minutes prior to the deadline.

Such failures were distributed unevenly among push and pull class users: 70% of these failed tasks were accepted by push class users. Our conjecture is that as push class users tend to accept tasks in advance (as observed in the previous section), future changes in their travel plans might render some tasks infeasible, thus causing them to ignore these tasks. Pull class users have significantly less such cases most likely because they only choose tasks when they are close to the task locations.

CONCLUSIONS AND FUTURE WORK

While mobile crowd-tasking is an attractive paradigm for distributed and collaborative execution of location-centric urban tasks, there is a dearth of testbeds that allow us to experimentally understand how humans respond to changes in various parameters, such as task recommendation or task pricing strategies. To address this limitation, we have built a large-scale crowd-tasking platform (front-ended by a mobile App called *TA$Ker*) for our university campus, where participating students are rewarded for performing a variety of campus-centric crowd-sourced tasks. We have described how we use an underlying indoor location tracking infrastructure to develop medium-grained predictions about individual-level movements on the campus, and use such predictions to support an advanced task recommendation strategy (which seeks to allocate tasks so as to minimize expected travel detours) in addition to the more conventional pull-based approach.

TA$Ker was deployed to a pool of 80 student volunteers, who performed around 800 total tasks over an initial deployment



(a) Push-class users.

(b) Pull-class users.

Figure 11. Detour efficiency of (a) push-class and (b) pull-class users.

period of 4 weeks. Our empirical results demonstrate a few key results:

- The push-based crowd-tasking model outperforms the conventional pull-based approach, resulting in not only a higher (and statistically significant) task completion rate, but also an approx. 50% lower average detour overhead. This is in spite of the 6-8 meter error in the underlying location technology, and the 68% accuracy of movement prediction observed on the campus testbed.
- Workers seem to prefer tasks that are simple and easy to execute (e.g., selecting one from a pre-defined set of responses) over slightly more complex tasks (e.g., using the smartphone camera to capture an image), even if the reduction in task execution time is less than the associated increase in the detour overhead.
- Providing individualized recommendations that utilize predicted travel patterns has an additional benefit (even though workers are themselves unaware that the recommended tasks are being personalized to their travel patterns): workers tend to accept such tasks earlier (as opposed to a more opportunistic strategy where they accept tasks only when they are near a task location) and perform them later (but within the specified task completion limit). Such behavior provides a platform with better lookahead on the tasks that are unassigned, which may then need additional incentives.

In ongoing and future work, we shall systematically study the impact of different task reward structures and incentive strategies (ranging from conventional micro-payment models to group-oriented, macro-reward models).

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 *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI '10)*.

2. 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.

3. Cen Chen, Shih-Fen Cheng, Hoong Chuin Lau, and Archan Misra. 2015. Towards city-scale mobile crowdsourcing: Task recommendations under trajectory uncertainties. In *Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*.

4. Marshall L Fisher. 1985. An applications oriented guide to Lagrangian relaxation. *Interfaces* 15, 2 (1985), 10–21.

5. Brent Hecht Jacob Thebault-Spieker, Loren Terveen. 2015. Avoiding the South Side and the Suburbs: The Geography of Mobile Crowdsourcing Markets *(CSCW '15)*.

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

7. Azeem J. Khan, Vikash Ranjan, Trung-Tuan Luong, Rajesh Krishna Balan, and Archan Misra. 2013. Experiences with performance tradeoffs in practical, continuous indoor localization.. In *IEEE WOWMOM*.

8. Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. 2013. The Future of Crowd Work. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13)*.

9. Nicolas Kokkalis, Thomas Köhn, Johannes Huebner, Moontae Lee, Florian Schulze, and Scott R Klemmer. 2013. TaskGenies: Automatically Providing Action Plans Helps People Complete Tasks. *ACM Transactions on Computer-Human Interaction* 20, 5 (2013), 27.

10. Archan Misra and Rajesh Krishna Balan. 2013. LiveLabs: Initial Reflections on Building a Large-scale Mobile Behavioral Experimentation Testbed. *SIGMOBILE Mobile Computing and Communications Review* 17, 4 (2013), 47–59.

11. Mohamed Musthag and Deepak Ganesan. 2013. Labor dynamics in a mobile micro-task market. In *SIGCHI*

Conference on Human Factors in Computing Systems. 641–650.

12. Kartik Talamadupula, Subbarao Kambhampati, Yuheng Hu, Tuan Anh Nguyen, and Hankz Hankui Zhuo. 2013. Herding the crowd: Automated planning for crowdsourced planning. In *1st AAAI Conference on Human Computation and Crowdsourcing*. 70–71.

13. 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 (CSDM)*.