MobiCrowd: Mobile Crowdsourcing on Location-based Social Networks

Yulong Tian*, Wei Wei†, Qun Li†, Fengyuan Xu*, Sheng Zhong*
*State Key Laboratory for Novel Software Technology, Nanjing University, China
† Department of Computer Science, College of William and Mary, USA

***Abstract*—The great potential of mobile crowdsourcing has started to attract attention of both industries and the research community. However, current commercial mobile crowdsourcing marketplaces are unsatisfactory because of the limited worker base and functionality. In this paper, we first revisit the foundation of performing mobile crowdsourcing on location-based social networks (LBSNs) through specially designed survey studies and comparison experiments involving hundreds of users. Our results reveal that active check-ins are good indicators of picking a right user to perform tasks, and LBSN could be an ideal platform for mobile crowdsourcing given proper services provided. We then propose both the centralized and decentralized design of MobiCrowd, a mobile crowdsourcing service built on LBSNs. Our evaluation, through trace-driven simulation and real-world experiments, demonstrates that the proposed schemes can effectively find workers for mobile crowdsourcing tasks associated with different venues by analyzing their location check-in histories.**

I. Introduction

With the prosperity of smartphones, the mobile crowdsourcing has become increasingly popular and attracted much attention. It is easy to participate since performing tasks are just at users' fingertips. More importantly, it enables new useful task categories, such as photography, location-aware surveys, service assessment, data collection, mobile sensing, price checks, deliveries, and property evaluation. For example, users can be rewarded to take photos of venues, answer location-aware queries, review services provided by businesses, participate in mobile sensing projects, check traffic of a road segment, share rides, or deliver items to a target location. Different from their web-based counterparts, these mobile crowdsourcing tasks share a universal feature that they are all centered on users' spatial locations. As we can see, mobile crowdsourcing extends web-based crowdsourcing from the digital domain to the physical world, which truly unleashes the enormous power of crowdsourcing.

Having sensed the great potential of mobile crowdsourcing, several commercial instances of smartphone-based crowdsourcing marketplaces have emerged [15]. However, these marketplaces are unsatisfactory in growth in both the size of worker base and their functionality. As shown in [15], over 400 days of activity, only several thousand users have successfully completed at least one task on a typical smartphone-based crowdsourcing marketplace. Considering the vast number of target locations the task requesters may potentially be interested in, the small worker base is far from being able to meet the demand for conducting mobile crowdsourcing everywhere. Moreover, these smartphone-based crowdsourcing marketplaces are constrained by their functionality, which are inherently the mobile equivalent of the web-based crowdsourcing marketplaces with limited smartphone-enabled permissions. As a result, users' contextual information (e.g., location, time, mobility) is not fully utilized to facilitate mobile crowdsourcing.

A previously unnoticed platform that can benefit mobile crowdsourcing is the location based social networks (LBSNs). Compared with traditional online social networks, LBSNs take a step further in that they provide the location-based features. Users of LBSNs can check-in at different venues (e.g., airports, restaurants), and these check-ins are broadcast to their friends. In this way, users share information about the places they visited. The past few years have seen soaring growth in LBSNs. For example, Foursquare, an LBSN founded in 2009, has over 50 million active users and 10 billion check-ins as of July 2017, with millions of new check-ins everyday [1]. The major online social networks have also provided new location-based features, such as Facebook Place and Twitter Place, which allow users to check-in and share check-ins with friends. These check-ins, combined with the online friendship connections revealed through the social graphs, provide an unprecedented opportunity to study human socio-spatial behaviors based on large-scale voluntarily contributed data. As existing location-sharing and social-networking platforms, LBSNs inherently have the culture of sharing and participation. The tens of millions of users on LBSNs further provide a tremendous potential work force to solve mobile crowdsourcing tasks. In addition, LBSNs maintain information about a vast number of venues around the world, as well as the users having visited these venues. This makes them the "encyclopedia" of places in the world people may be interested in. Combining all these unique features together makes LBSNs the ideal platform for mobile crowdsourcing.

In this paper, we propose MobiCrowd, a mobile crowdsourcing service built over LBSNs. we first revisit the foundation of performing mobile crowdsourcing on LBSNs through specially designed survey studies and comparison experiments conducted over Foursquare and MTurk involving hundreds of users. We discover that mobile crowdsourcing tasks can be performed significantly faster on LBSNs with much higher output quality (Section II). Inspired by our study results, we present both the centralized and decentralized design of MobiCrowd, which leverages the clustering phenomenon exhibited in users' check-in activities (Section III). Our schemes find suitable workers for mobile crowdsourcing tasks by taking into account both spatial and temporal factors, and they are not intrusive to LBSN users in that only users interested in participating in MobiCrowd can receive those tasks. The centralized scheme assumes full knowledge of user check-ins on LBSN. When this is not the case, the decentralized scheme can be activated to run in a fully distributed fashion. A two-stage routing is introduced to speed up the task delivery. The effectiveness of the proposed schemes is verified through both trace-driven simulation and real-world experiments, compared with other four approaches (Section IV). The simulation using realistic user check-in traces shows that the test check-ins of the workers chosen by our schemes are close to the pending tasks both spatially and temporally, implying a high tendency to work on the tasks. The 49-day long real-world experiment demonstrates that our schemes can accurately find workers who are more likely to perform mobile crowdsourcing tasks

in practice.

II. Survey Study on Mobile Crowdsourcing

To investigate the feasibility of performing mobile crowdsourcing on LBSNs, we have conducted two surveys. The first survey was conducted on Foursquare, the most popular LBSN. In comparison, the second survey was launched on MTurk, the largest online crowdsourcing marketplace. Comparing results of the two surveys reveals two important findings: first, mobile crowdsourcing tasks are done much faster on LBSNs; second, task outputs have significantly higher quality on LBSNs.

A. Method

In this paper **workers** refer to the individuals who accept and perform crowdsourcing tasks. **Requesters** are the individuals or companies who post tasks on the crowdsourcing marketplaces. Based on the temporal requirement on when the workers should visit the target venue, mobile crowdsourcing tasks can be classified into two categories. Tasks falling into the first category only require that the workers have been to the target venue before, which we refer to as **past-presence tasks**. Examples of such tasks include location-aware queries/surveys, venue/service/product reviews, and recommendations. On the other hand, to perform tasks of the second category, workers need to visit the target venue in the future. We refer to these as **future-presence tasks**. As shown in a previous study [15], typical examples of this kind of tasks include mobile sensing, data collection (such as photography), and deliveries/rides. In our surveys, we use location-aware queries as instances of mobile crowdsourcing tasks to study the potential of performing mobile crowdsourcing on LBSNs. To eliminate bias incurred by self-composed queries, the queries used in the surveys are randomly sampled from Yahoo! Answers [2]. Each query is associated with a target venue. Participants of the first survey are all Foursquare users who have visited the target venues according to their tips left on Foursquare. Participants of the second survey are recruited on MTurk. The detailed design of the two surveys is presented below.

Design of the first survey. Foursquare does not allow unauthorized access to users' check-in information. However, besides check-ins, Foursquare also enables social recommendations through tips, which are a small snippet of text associated with a venue. Unlike check-ins are only accessible to friends, these tips are published on the pages of associated venues. Therefore, given the target venue of a location-aware query, to find Foursquare users who have visited that location, we can simply browse the venue page and sample users who have left tips there.

Foursquare does not provide direct communication mechanisms that we can use to send queries to the sampled users. However, we found that the majority of Foursquare users have their accounts linked with their Facebook accounts, and Facebook allows sending messages to strangers (by charging a small amount of money). Therefore, we sampled Foursquare users who have a linked Facebook account. In our survey we used 5 location-aware queries, and for each query we sampled 60 users, so in total we sent queries to 300 Foursquare users. The survey ran for five days and one query was sent out at noon each day.

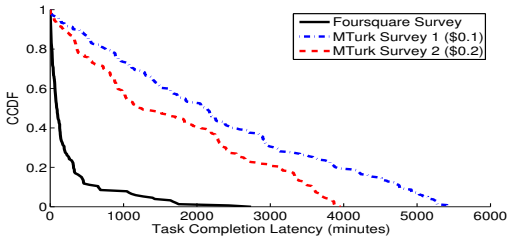

Fig. 1. CCDF of task completion latency

Design of the second survey. We also conducted a comparison survey on MTurk, which is a general crowdsourcing marketplace where requesters can post Human Intelligence Tasks (HITs) to be completed by workers. Each task normally pays workers a small amount of compensation. Previous research shows that 90% of the HITs pay less than $0.10 [11], and the majority of workers on MTurk come from the US and India [21].

We created 5 HITs on MTurk for the 5 location-aware queries used in the first survey. Each HIT asks workers to answer one query. To study the influence of compensation on tasks' completion speed, we posted each HIT twice. Once with a compensation of $0.10 and once with a compensation of $0.20, so the survey consists of two rounds. We first posted HITs with lower compensation. One week later we ended these HITs and posted HITs with higher compensation. In this way we prevent the high-payment HITs from interfering the completion of low-payment HITs. In order to be able to compare the completion speed with the first survey, we posted the HITs also at noon.

B. Results

Our first survey received 214 responses from the 300 queried Foursquare users, meaning that 71.3% of the queried users responded to the survey. The first round of our second survey ($0.10 reward) involved 411 responses, and the second round ($0.20 reward) involved 532 responses. The average working time for MTurk workers to finish the survey is 114 seconds, which yields an hourly wage of $3.16 and $6.32 respectively. Both are higher than the $2.00 average hourly wage of MTurk workers [15]. To investigate the performance of mobile crowdsourcing in these two scenarios, we focus on two metrics: task completion latency and response accuracy.

Task completion latency. Task completion latency is defined as the temporal interval between the time when a task is sent out (the first survey) or posted (the second survey) and when a response is received. Since the number of responses of the two surveys are not equal, to fairly compare completion latency of the two surveys, we only consider the completion latency of the first 214 responses received in each round of the second survey, because 214 responses were received in the first survey.

Figure 1 compares the CCDF (complementary cumulative distribution function) of completion latency of responses received in the first survey with those received in the second survey. The median task completion latency is 93 mins for the Foursquare survey, 1249 mins (20.8 hours) for the first 214 responses of the MTurk survey with a $0.2 reward, and 2137 mins (35.6 hours) for the first 214 responses of the MTurk survey with a $0.1 reward, demonstrating that the queries are

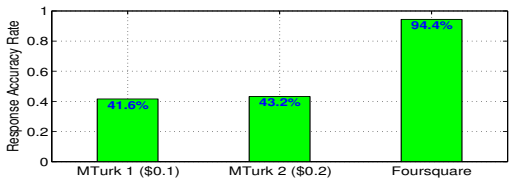

Fig. 2. Response accuracy rate

responded significantly faster on LBSNs. The cause of this difference is that as MTurk style markets grow rapidly in size, due to the lack of an effective task search/recommendation mechanism, workers on MTurk are overwhelmed by the vast number of open tasks, and it is extremely time-consuming for a worker to select a task he/she is suitable for [5]. As a result, workers spend a large amount of time searching for tasks and they still tend to make sub-optimal decisions to work on tasks they are actually not good at. In contrast, as we did in the first survey, mobile crowdsourcing tasks can be forwarded to the right LBSN users based on their visit histories. This push mechanism in the form of a recommendation engine helps eliminate task searching time, and also reduce the time spent on performing tasks. It is interesting to notice that the CCDF of task completion latency is approximately a straight line for both rounds of the MTurk survey, which implies that our tasks are completed at a constant rate on MTurk. This indicates that given a fixed reward for a task, there is a constant background probability that workers find and complete the task. This probability is probably related to workers' behavior on MTurk, e.g., searching for tasks by key words.

Response accuracy. All the location-aware queries used in our surveys require the respondents provide an explanation of their answer. A response is regarded as accurate as long as it is factually correct and its explanation matches the target venue. For example, one query in our surveys is "Which theme park in Disney World Orlando do you think is the best for adults to visit and why?". An accurate answer should be one of the four theme parks in Disney World with a reasonable explanation. As shown in Figure 2, the accuracy rate of our first survey is significantly higher than that of the second survey. Doubling compensation in the second survey does not effectively improve response accuracy, which is consistent with a previous finding that increased financial incentives only increase task completion speed, but not work quality [14].

We believe that this large gap of response accuracy is caused by the internal mechanisms of the two platforms. Workers on MTurk have an incentive to sacrifice accuracy in favor of finishing tasks quickly, since they mainly work for financial compensation. The faster they work, the more compensation they can get. Also, all the MTurk workers can accept and respond to our location-aware queries, no matter if they have visited the target venues or not. Due to the anonymity of the workers, requesters on MTurk have very little visibility into characteristics of the workers. In addition, response accuracy is further harmed by the fact that some workers have multiple accounts on MTurk, as we have received exactly the same responses to our queries. On the other hand, as existing social media platforms, LBSNs inherently have the culture of sharing and participation. Users participate LBSNs mainly for intrinsic motivations such as fun and social appreciation [12], and thus they lack the incentive to sacrifice response accuracy. Moreover, LBSNs record all the users' check-in information. Mobile crowdsourcing tasks can be forwarded to users who are most likely to work on them by analyzing the users' previous visits. In our case, we send location-aware queries to users who have visited the target venues. This "targeting" of tasks is apparently advantageous to improving work quality of mobile crowdsourcing.

C. Discussion

Our survey results demonstrate that mobile crowdsourcing conducted over LBSNs significantly outperforms that conducted on traditional crowdsourcing marketplaces in both task completion speed and output quality, even traditional crowdsourcing marketplace have considerable incentives. The low performance of the latter is attributed to the internal limitations imposed by those platforms. By contrast, mobile crowdsourcing can leverage features of LBSNs to achieve both low completion latency and high work quality. This implies that LBSNs have great potential to serve as the platform of mobile crowdsourcing. In the next section, we will describe how to build a mobile crowdsourcing service based on existing LBSNs with both a centralized design and a decentralized design.

III. MobiCrowd Design

In this section we present the design of MobiCrowd, a mobile crowdsourcing service built over LBSNs. MobiCrowd consists of a centralized scheme and a decentralized scheme. The former assumes full knowledge of user check-ins on LBSNs, while the latter does not and all the operations are with respect to individual LBSN users. The details of both schemes are presented below.

A. Centralized Task Recommendation

1) Motivation: The intuitive way is to build a central platform where all the LBSN users can post tasks and search for tasks he/she is willing to work on. However, as shown in our survey study and some previous work [5], [11], [28], this approach is highly inefficient. Beyond a pull mechanism similar to MTurk, a push mechanism in the form of a task recommendation engine boasts several benefits. For instance, workers will receive tasks they are more likely to work on, which saves their efforts that would otherwise be spent on searching the task marketplace.

The intuition behind our task recommendation algorithm is that by extracting and analyzing the candidates' active regions, we are able to find workers suitable for the target task both spatially and temporally. To achieve this our scheme analyzes both spatial proximity and temporal activeness of the candidates, whose details are presented below.

2) Scheme Design: ***Candidate selection.*** To find suitable workers for a mobile crowdsourcing task, it is inefficient to investigate all the registered workers on an LBSN, which usually has millions of users. Therefore, our task recommendation scheme starts by selecting a subset of workers as candidate workers, whose check-ins will be analyzed later. We leverage an LBSN feature that for each venue, LBSNs maintain the list of users that have checked-in there. The candidate selection process first finds venues that are within a range δ from the target venue (including the target venue itself). All the workers

Algorithm 1 Spectral Clustering

Require: The set of venues visited by a candidate, $v_1, ..., v_m$
1: **for** $i = 1$ to $m - 1$ **do**
2: **for** $j = i + 1$ to m **do**
3: $distanceVector.push\_back(d(v_i, v_j))$
4: Sort all the values in $distanceVector$ in increasing order
5: $\sigma = distanceVector.get(10\% * distanceVector.size)$
6: Construct a $m * m$ similarity matrix W, where $W_{ij} = e^{-\frac{(d(v_i,v_j))^2}{2\sigma^2}}$
7: Compute D as the diagonal degree matrix of W
8: Compute the normalized Laplacian $L_{rw} = I - D^{-1}W$, where I is the identity matrix
9: Compute the eigenvalues and eigenvectors of L_{rw}
10: Assume the difference between two eigenvalues λ_r, λ_{r+1} is the largest among all the pairs of consecutive eigenvalues
11: $k = r$
12: Let $x_1, ..., x_k$ be the first k eigenvectors of L_{rw}, and F be an $m * k$ matrix with x_i as its columns
13: Let $y_1, ..., y_m$ be the rows of F. Cluster $y_1, ..., y_m$ into clusters $C_1, ..., C_k$ with k-means algorithm
14: return $C_1, ..., C_k$

that have checked-in at these venues before are treated as candidates.

Spectral clustering. The second step is to group the venues visited by each candidate into clusters. Venues in the same cluster are close to each other, while they are relatively far away from venues in other clusters. For the sake of efficiency and quality, we leverage the spectral clustering technique [22], which is well studied and popular in practice due to its excellence compared to other traditional algorithm such as K-means [13]. As shown in Algorithm 1, assuming a candidate has visited m venues $v_1, ..., v_m$, the input to spectral clustering is a $m*m$ similarity matrix W, where W_{ij} reflects the closeness between v_i and v_j. We compute W_{ij} using the Gaussian similarity function defined as $e^{-\frac{(d(v_i,v_j))^2}{2\sigma^2}}$, where $d(v_i, v_j)$ is the great-circle distance between v_i and v_j. Previous work have shown that spectral clustering results are sensitive to the choice of σ, which controls decay rapidity of the similarity measure [13], [22], [26]. In our case, since different candidates have different activity patterns, rather than using a constant σ for all the candidates, the choice of σ should depend on each candidate's own check-in history. To determine σ for a candidate, we compute the distance between every pair of venues checked-in by this candidate, and sort all these distances in increasing order. σ is set to the 10th percentile distance in this sorted list. Similar methods have been used in previous work [22], and our evaluation shows that with σ determined in this way, spectral clustering produces high quality results. Another parameter used in spectral clustering is k, the number of output clusters. To decide this parameter we adopt the eigengap heuristic, which states that k should be set equal to the rank of the eigenvalue where the largest difference between two consecutive eigenvalues appears [13]. Spectral clustering first performs a non-linear dimension reduction on the input matrix, and then a k-means algorithm is applied to the low-dimensional embedding. The output is the clusters of venues visited by the candidate.

Candidate ranking. Given the spectral clustering results, the third step is to assign each candidate a ranking score by tasking into account both spatial proximity and temporal activeness of the candidate. The top-N highest ranked candidates are returned as chosen workers to whom we forward the mobile crowdsourcing task. Inspired by our survey, we think that workers who are more active recently and whose check-ins are closer to the target venue are more likely to work on the task. The way to compute the ranking score of a candidate is discribed in Algorithm 2, which integrates the temporal activeness and the spatial proximity. Assume the venues visited by the candidate are grouped into k clusters. We first assign a temporal weight to each cluster, which is the sum of temporal weights of all the candidate's check-ins associated with venues in this cluster. The temporal weight of each check-in is decided by an exponential decay function defined as $e^{-\alpha\frac{T-t_j}{T-t_0}}$, where t_j is the timestamp of this check-in, t_0 is the timestamp of the candidate's first check-in, T is the current timestamp, and α is a scaling coefficient controlling the decay rate. With this exponential time-decay function, the relevance of older check-ins is reduced, and more recent ones promptly become more prominent in computing temporal weight. Similar time-decay functions have been adopted by previous work [7], [16].

We sort all the clusters by their temporal weights in decreasing order. A cluster with a higher weight means it is more temporally important. The reason may be that it has more check-ins, or its check-ins occur more recently, or both. Starting from the highest-weighted cluster, we first compute the minimum bounding rectangle (MBR) that covers all the venues in this cluster. If the target venue is within the MBR, then the ranking score of the candidate is added by the temporal weight of this cluster. Otherwise, we compute the GPS coordinates of the temporally adjusted centroid of this cluster, whose x coordinate is defined as $\sum_{c_i \in C} c_i.xcoord\frac{c_i.weight}{C.weight}$, where c_i is a check-in associated with a venue in this cluster. The y coordinate is defined accordingly. It is easy to see that the geographic location of this centroid is adjusted by the temporal weights of check-ins falling into the cluster. More recent check-ins have a larger impact on the location of the centroid. Assume d is the distance between the temporally adjusted centroid and the target venue. The probability that the candidate travels a distance larger than d is approximated by the ratio of displacements between consecutive check-ins made by the candidate that are larger than d. The ranking score of the candidate is added by the temporal weight of the current cluster multiplied by this probability. As shown in Algorithm 2, this process is repeated until the relative weight of the current cluster compared to all the clusters is smaller than a threshold τ, which means that we only consider the temporally significant clusters when computing a candidate's ranking score.

The top-N ranked candidates are returned as output of our task recommendation scheme, to whom the mobile crowdsourcing task is forwarded. Once the task is finished by a worker, the reply is sent back to the task requester. As a design option, a check-in at the target venue can be associated with the reply as a proof that the worker has visited target venue.

B. Decentralized Task Routing

1) Motivation: The centralized scheme leverages full knowledge of LBSN users' check-ins to choose workers for mobile crowdsourcing tasks, and thus it assumes support from LBSNs. Sometimes, however, such support is not available. For example, suppose a third party or a group of LBSN users would like to build the mobile crowdsourcing service. It is unlikely that the LBSNs will sacrifice their commercial

Algorithm 2 Candidate Scoring

Require: Spectral clustering results of a candidate, $C_1, ..., C_k$

Sort all the check-ins by their timestamps in increasing order. Assume t_0 is the timestamp of the candidate's first check-in
for $i = 1$ to k **do**
  Assume n of the candidate's check-ins $c_1, ..., c_n$ are associated with venues in cluster C_i
  for $j = 1$ to n **do**
    $C_i.weight = C_i.weight + e^{-\alpha \frac{T-t_j}{T-t_0}}$, where T is the current timestamp, t_j is the timestamp of c_j
  $totalWeight = totalWeight + C_i.weight$
Sort all the clusters by their temporal weights in decreasing order. Let the sorted list be $C'_1, ..., C'_k$
for $i = 1$ to k **do**
  if $C'_i.weight/totalWeight < \tau$ **then**
    break
  if the target venue is within the MBR of C'_i **then**
    $rankingScore = rankingScore + C'_i.weight$
  else
    Compute (x_i, y_i) as the temporally adjusted centroid of cluster C'_i
    $distance = d((x_i, y_i), (x_t, y_t))$, where (x_t, y_t) is the coordinates of the target venue
    Compute p as the approximate probability that the candidate travels a distance larger than $distance$
    $rankingScore = rankingScore + p * C'_i.weight$
return $rankingScore$

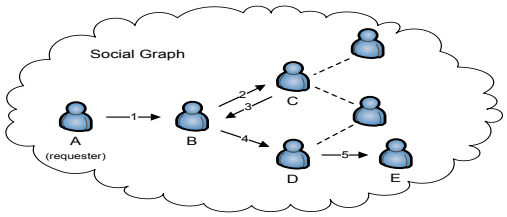

Fig. 3. A task routing path with $TTL = 5$

interests and allow outsiders to access their central database. In this scenario, a decentralized scheme that does not require global knowledge of LBSNs is needed. This scheme should be able to route mobile crowdsourcing tasks to suitable workers based only on local knowledge available to individual users.

We assume a peer-to-peer architecture, in which each node (a software agent running on mobile devices/computers) is associated with a participant of the mobile crowdsourcing service. Two nodes can communicate with each other if the corresponding users are friends on the LBSN, in which case we say the nodes are linked and they are neighbours on the friendship graph. We also assume that a node can access the full check-in histories of its neighbours. This is a rational assumption as sharing check-ins between friends is a basic function provided by LBSNs. Based on the type of pending mobile crowdsourcing tasks, the problem of decentralized task routing can be separated into two cases, past-presence task routing and future-presence task routing. The latter is obviously more difficult to solve, since both spatial and temporal factors need to be considered in order to route tasks to workers who are most likely to work on them within a short time period in the future. In this subsection we first present our routing scheme for future-presence tasks. The routing scheme for past-presence tasks is briefly discussed thereafter.

2) Scheme Design: ***Future-presence tasks.*** Our solution for future-presence tasks is essentially a two-stage geographic routing scheme. It shares the same intuition with our centralized scheme that workers who are spatially closer to the target venue and who are more temporally active recently are more likely to work on the task. At the first stage routing

Algorithm 3 Neighbour Scoring

Require: A node u holds a task with target venue v_t

Assume $u_1, ..., u_n$ are the n neighbours of u that haven't received the task before
for $i = 1$ to n **do**
  Group the venues visited by u_i into k clusters $C_1, ..., C_k$ with spectral clustering (see Alg. 1)
  Compute the temporal weight of each cluster. Assume $totalWeight$ is the sum of weight of all the clusters
  Sort all the clusters by their temporal weight in decreasing order. Let the sorted list be $C'_1, ..., C'_k$
  for $j = 1$ to k **do**
    if $C'_j.weight/totalWeight < \tau$ **then**
      break
    $majorWeight = majorWeight + C'_j.weight$
  for $j = 1$ to k **do**
    if $C'_j.weight/totalWeight < \tau$ **then**
      break
    Compute (x_j, y_j) as the temporally adjusted centroid of cluster C'_j
    $distance = d((x_j, y_j), (x_t, y_t))$, where (x_t, y_t) is the coordinates of v_t
    $u_i.distance = u_i.distance + distance \frac{C'_j.weight}{majorWeight}$
    if $distance < \gamma$ **then**
      if v_t is within the MBR of cluster C'_j **then**
        $u_i.score = u_i.score + C'_j.weight$
      else
        Compute p as the approximate probability that u_i travels a distance larger than $distance$
        $u_i.score = u_i.score + p * C'_j.weight$
  return $u_i.distance$ and $u_i.score$

Algorithm 4 Task Forwarding

Require: $u_i.distance$ and $u_i.score$ of the n neighbours $u_1, ..., u_n$ of u, where u is the node holding the task, $u_1, ..., u_n$ haven't received the task before

for $i = 1$ to n **do**
  if $u_i.score \neq 0$ **then**
    $U_2 = U_2 \cup u_i$
  $U_1 = U_1 \cup u_i$
if $u.score \neq 0$ **then**
  if $U_2 \neq \varnothing$ **then**
    Pick u_s as the node in U_2 with the highest $score$
    if $u_s.score > u.score$ **then**
      Forward the task to u_s
    Forward the task back to the neighbour it was received from
  else
    Forward the task back to the neighbour it was received from
else
  if $U_2 \neq \varnothing$ **then**
    Forward the task to the node in U_2 with the highest $score$
  else
    Pick u'_s as the node in U_1 with the smallest $distance$
    if $u'_s.distance < u.distance$ **then**
      Forward the task to u'_s
    else
      Forward the task back to the neighbour it was received from

decision is made solely based on geographic proximity, while at the second stage both geographic proximity and temporal activeness are considered when deciding the next hop. When a task requester has a pending task, the node initiates a task routing path on the social graph starting from each of its neighbours. At each step of a path, the task is forwarded following a greedy routing algorithm described below, and back-tracking is used when no better neighbour can be found. The maximum number of hops a task can be forwarded is determined by a parameter TTL, which counts both advancing and back-tracking steps. Figure 3 shows an example task routing path with $TTL = 5$, where hop 3 is the back-tracking. All the nodes traversed by the paths are candidate workers who can work on the task. Once a result is obtained, it is sent back along the same path to the task requester. Similar to the centralized case, a check-in at the target venue can be provided as a proof that the worker has visited the target venue.

At each hop of a task routing path, our scheme computes two values for each neighbour that has not received the task from the current node before, including a weighted average distance to the target venue and a ranking score. The two-stage routing decision will be made based on these values. Algorithm 3 describes how to derive the two values of each neighbour. We only consider temporally significant clusters, which are the clusters whose relative temporal weights compared to all the clusters are larger than τ. The weighted average distance is the average of distances from the temporally significant clusters to the target venue, weighted by the temporal weight of each cluster. The ranking score is derived in the same way as that in the centralized scheme for those temporally significant clusters within a distance of γ to the target venue. The reason why the routing decision needs to be made based on these two values of each neighbour is that at the beginning hops of a task routing path, almost all the neighbours are far from the target venue, which usually leads to a ranking score of zero, as it is unlikely that their check-in histories contain any large displacement. Therefore, it is infeasible to use ranking scores as the single criterion for route selection. Our solution is, at the early stage of task routing, only spatial proximity is considered to choose the next hop. When the task routing path has reached nodes with neighbours close to the target venue (determined by γ), ranking scores that consider both spatial proximity and temporal activeness are used to select a neighbour to forward the task.

Algorithm 4 presents how to decide the next hop of the task routing path based on the outputs of Algorithm 3. Let U_1 be the set of neighbours which have not received the task from the current node u before, and U_2 be the set of nodes in U_1 whose ranking scores are nonzero. Assume $u.score$ is the ranking score of u. If $u.score$ is nonzero, the task is forwarded to the highest-scored node in U_2 whose ranking score is larger than $u.score$, since this node is the "closest" to the target venue. If no node in U_2 has a ranking score larger than $u.score$, the task is forwarded back to the neighbour from which it was received, i.e., back-tracking when no closer neighbour can be found. If $u.score$ is zero, we regard any neighbour with a nonzero ranking score as closer to the target venue. Thus, the task is forwarded to the highest-scored node in U_2 if U_2 is not empty. If U_2 is empty, then task routing falls back to relying solely on geographic proximity. In this case, the task is forwarded to the node in U_1 with the smallest weighted average distance to the target venue, as long as this distance is smaller than u's weighted average distance to the target venue. If no node in U_1 is closer to the target venue compared with u, then back-tracking is performed and the task is sent back to the neighbour it was received from. And in addition, once a node receive a task due to back-tracking, if the value of TTL doesn't exceed limitation, the current node will again perform Algorithm 4 (notice that the set U_1 is different from the former time).

Past-presence tasks. The problem of past-presence task routing is much simpler to solve, as the tasks only need to be routed to workers who have visited the target venues before, and there is no need to predict user movement in the future. To save space we briefly introduce our solution. At each hop of a task routing path, if multiple neighbours of the current node have checked-in at the target venue previously, the task is forwarded to the one with the most recent check-in at the target venue. Otherwise, if no neighbour has been to the target venue before, the task is forwarded to the neighbour closest to the target venue, as long as this neighbour is closer to the target venue compared with the current node. Here, closeness is defined as the minimum distance between the temporally significant clusters of a node and the target venue. If compared with the current node no closer neighbour can be found, the task is forwarded back to the neighbour it was received from.

TABLE I. STATISTICS OF THE DATASETS

Dataset	Users	Edges	Check-ins	Duration
Foursquare1	18,107	115,574	1,622,526	15 months
Foursquare2	93,115	NA	7,956,679	4 months
Gowalla	196,591	950,327	3,674,591	20 months
Brightkite	58,228	214,078	2,920,919	30 months

C. Incentive Mechanisms

In this subsection we discuss the incentive mechanisms that can be designed for mobile-crowdsourcing services built over LBSNs. To engage participants, mobile crowdsourcing can leverage both intrinsic and extrinsic motivations. The former includes game-based features and social psychological incentives, and the later generally refers to financial incentives.

As shown in previous work, one of the major reasons for which people join LBSNs is for fun [12]. Therefore, the mobile crowdsourcing services can provide games like one-day-mayorship to entertain users and engage them to continuously perform tasks. Previous research shows that social incentives, such as altruism, feeling good being helpful to others, and social reciprocity, indeed improve the work quality of crowdsourcing [20]. LBSNs inherently have the culture of sharing and participation, which makes social incentives the ideal motivational approach that can be used by mobile crowdsourcing built over LBSNs. The most straightforward approach to motivating participation is through financial incentives. However, such extrinsic motivations should be used with care. It has been shown in previous work that increased financial incentives only increase work completion speed, but not work quality [14], [20], which has also been verified by our survey study. Therefore, instead of being used as the single motivation, financial incentives should be used in conjunction with intrinsic incentives.

IV. EVALUATION

We evaluate the effectiveness of our proposed schemes through both trace-driven simulation and real-world experiments. The four LBSNs datasets used in our evaluation include *Foursquare1* [9], *Foursquare2* [4], *Gowalla* [6], and *Brightkite* [6]. We consider the check-ins whose locations are within the mainland of the US, where the four datasets have the majority of check-ins. The statistics of the resulting datasets are shown in Table I. To save space, we only present the evaluation results obtained with the *Foursquare1* dataset below, the results obtained using other datasets are very similar.

A. Spectral Clustering Results

Both our centralized and decentralized schemes leverage spectral clustering to group the venues visited by each user into clusters, and thus their performance relies on the clustering results. Our spectral clustering algorithm (Algorithm 1) adjusts parameter σ in the Gaussian similarity function based

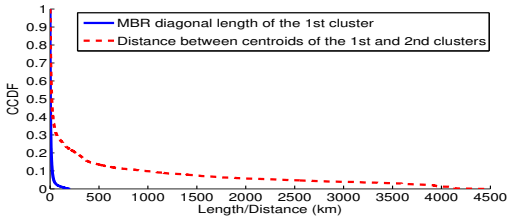

Fig. 4. CCDF of MBR diagonal length of the most temporally significant cluster, and CCDF of the distance between centroids of the two most temporally significant clusters of each user

on each user's own check-in history, and it determines the optimal number of clusters using the eigengap heuristic [13]. Evaluation results show that our algorithm can derive fine-grained clustering information form user check-ins. As an example, we plot in Figure 4 the CCDF of the diagonal length of the minimum bounding rectangle (MBR) of each user's most temporally significant cluster, as well as the CCDF of the distance between the centroids of the two most temporally significant clusters of each user. The figure illustrates that venues in the same cluster are close to each, and they are far away from venues in other clusters. This also serves as a real-world evidence that user check-ins on LBSNs indeed exhibit the clustering phenomenon.

B. Evaluation of the Centralized Task Recommendation Scheme

1) Simulation: We evaluate our centralized task recommendation scheme through both simulation and real-world experiments. Our evaluation focuses on future-presence tasks. The simulation is conducted using realistic user check-in traces in the LBSN dataset. In each run of the simulation, a random timestamp within the temporal duration of the LBSN dataset is chosen as the task initiating time, and a random venue is selected as the target venue of the task. All the user check-ins before this timestamp are treated as check-in histories available to our task recommendation scheme and the baseline schemes. We use each user's check-in immediately after the timestamp as the test data to measure his/her tendency to work on the task, and we consider two measures, *spatial closeness* and *temporal closeness*. The former is defined as the geographic distance between the test check-in and the target venue, and the latter is defined as the temporal interval between the timestamp of the test check-in and the task initiating time. Our intuition is that the smaller spatial closeness and temporal closeness are, with higher tendency the user will work on the task, as his/her test check-in matches the simulated task both spatially and temporally. In the simulation we investigate the spatial closeness and temporal closeness of the workers chosen by our scheme and those chosen by the baseline schemes.

Spatial closeness. We compare our task recommendation scheme with four baseline schemes listed bellow. Each baseline scheme ranks the candidate workers with a non-trivial method, and the top-N ranked candidates are returned. Notice that the candidate workers are the users who have check-ins within a range of δ from the target venue in their check-in histories.

Most Recent (MR): Each candidate is ranked by the distance from the latest check-in in her check-in history to the target venue in increasing order.

Most Frequent (MF): Each candidate is ranked by the distance from her most frequently visited venue to the target venue in increasing order.

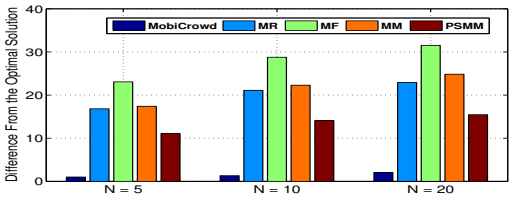

Fig. 5. *DFO* of different task recommendation schemes, normalized by *DFO(MobiCrowd)* with $N = 5$

Order-1 Markov Model (MM): The next check-in location of each candidate is predicted using the Order-1 Markov Model [23]. The candidates are ranked by the distance from the predicted location to the target venue in increasing order.

Periodic & Social Mobility Model (PSMM): The next check-in location of each candidate is predicted using the PSMM model proposed by Cho et al. [6]. Since this model requires the timestamp of the predicted check-in as input, we feed it with the timestamp of the test check-in of each candidate. Note this "future" information is unavailable in practice. The candidates are ranked by the distance from the predicted location to the target venue.

In order to compare the output quality of different schemes in terms of spatial closeness to the target venue, we first compute the optimal solution as the N candidates whose test check-ins are the closest to the target venue, based on the test check-in data not available to the recommendation schemes. Assume $d_1, ..., d_N$ are the geographic distances from the test check-ins of these N candidates to the target venue, which are sorted in increasing order, and $s_1, ..., s_N$ are the distances from the test check-ins of the N candidates chosen by a recommendation scheme S to the target venue, which are also sorted in increasing order. The *Difference From Optimal (DFO)* metric is defined as $DFO(S) = \sqrt{\frac{\sum_{i=1}^{N}(d_i - s_i)^2}{N}}$. *DFO* measures the consistence between the solution returned by a task recommendation scheme and the optimal solution: the smaller *DFO* is, the more consistent the returned solution is to the optimal.

The parameters used in our scheme are set as: $\alpha = 10, \tau = 0.1, \delta$ = 10 km. In each run of the simulation a simulated task is generated randomly, and N, the number of chosen workers, is set to 5, 10, 20, respectively. We analyze users who have at least 30 check-ins before the task initiating time, and at least 1 check-in after, using the five task recommendation schemes, and record the obtained *DFO* values. Figure 5 compares the average *DFO* value of our scheme (*MobiCrowd*) with those of the four baseline schemes over 5,000 runs of the simulation, which shows that the solution returned by our scheme matches the optimal solution significantly better than those returned by the baseline schemes. This demonstrates that given a mobile crowdsourcing task, our scheme is able to find more suitable workers in terms of spatial closeness to the target venue.

Temporal closeness. Figure 6 compares the temporal closeness of solutions returned by our scheme with those returned by the baseline schemes. The average temporal interval between the test check-ins of the candidates chosen by our scheme and the task initiating time is less than 1 day over 5,000 runs of the simulation, while it is larger than 4 days for the

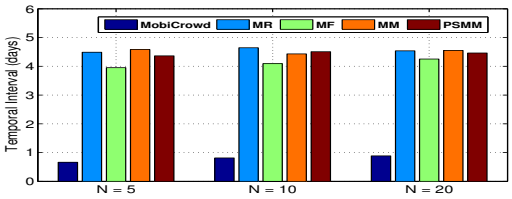

Fig. 6. Average temporal interval between test check-ins of chosen candidates and task initiating time

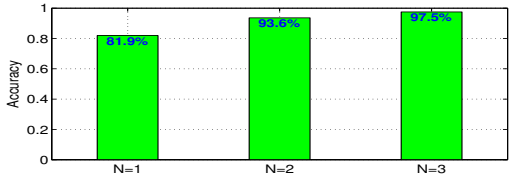

Fig. 7. Accuracy of our task recommendation scheme

baseline schemes. This verifies our assumption that users' past activeness is a good indicator for their future activeness. Our scheme ranks candidate workers by taking into consider both spatial proximity and temporal activeness of their check-ins, and thus the chosen workers are more active in the near future compared with the workers chosen by the baseline schemes.

2) Real-world Experiments: The performance of our task recommendation scheme is also evaluated through real-world experiments. We implement a smartphone App for participants of experiments to download and use. Participants can use this App to check-in anytime, perform tasks when they are at a target venue, download new tasks and upload their check-ins and completed tasks to our server when Internet access is available. During the experiment we post 5 new tasks on the server at the midnight of each day. Each task is associated with a target venue in the region participants live, and the venue is randomly sampled from Foursquare. To complete a task the participants need to go to the target venue and take a picture there. When a participant downloads new tasks, our application pulls the most recent 5 tasks from the server, and it always shows the recently downloaded 5 tasks. When a participant uploads completed tasks, the GPS coordinates of where he/she performs the tasks are uploaded, with which we can verify if the participants actually go to the target venues.

The experiment recruited 14 participants. They were told to act according to their own willing, interests and convenience. They would be rewarded with a small amount of financial incentive for each completion of a task. During 49 days of running the experiment, 245 tasks were posted on the server, among which 204 were completed. The unfinished tasks were all far away from daily check-ins of the participants. In total we received 1422 check-ins from the participants, which we use to evaluate the performance of our task recommendation scheme. We define the *accuracy* metric as the experiment probability that the participant who first completes a task is among the N workers chosen by our scheme. Figure 7 shows the accuracy of our scheme with $N = 1, 2, 3$. The results demonstrate that our scheme can effectively find workers who are more likely to work on mobile crowdsourcing tasks in practice.

C. Evaluation of the Decentralized Task Routing Scheme

The performance of our decentralized task routing scheme is evaluated through simulation. In each run of the simulation a task is generated with a random timestamp and a random

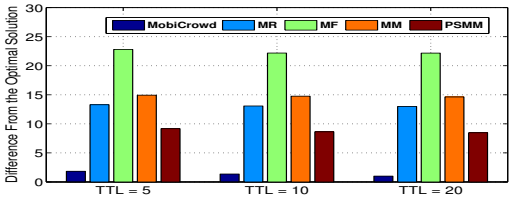

Fig. 8. *DFO* of different task routing schemes, normalized by *DFO(MobiCrowd)* with $TTL = 20$

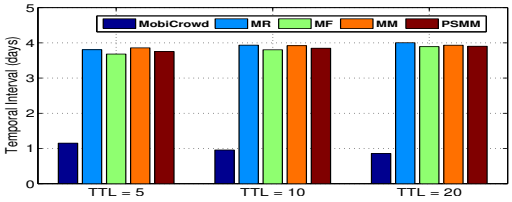

Fig. 9. Average temporal interval between test check-ins of top-ranked traversed nodes and task initiating time

target venue, and a node in the LBSN social graph is randomly chosen as the task requester. The requester initiates a task routing path from each of its neighbours with maximum length *TTL*. At each hop of a task routing path, the next step is chosen from the neighbours of the current node following our scheme proposed in Section III-B. Therefore, assuming the requester has n neighbours, at most $nTTL$ nodes will be traversed by the task routing paths. All the traversed nodes are candidate workers who can work on the task. The parameters used in our scheme are set as: $\alpha = 10, \tau = 0.1, \gamma$ = 100 km. Notice that this simulation is not a real-world experiment studying human behaviors. Instead, it serves as an investigation of what would be possible for such a routing scheme, i.e., whether the task routing paths can reach nodes whose test check-ins are close to the pending task both spatially and temporally. By simulating task forwarding on a real-world LBSN social graph, we can investigate the performance of routing schemes without relying on the voluntary participation of users. In practice some incentive mechanisms can be used to motivate task forwarding. For example, one possible scheme is that all the users on the path leading to the task accomplisher can share the reward [19].

In the simulation, we compare our scheme with four decentralized baseline schemes. The difference between our scheme and the baseline schemes is that different routing criterions are used when deciding the next step at each hop. Our scheme ranks neighbours of the current node based on their ranking scores derived by Algorithm 3, while the baseline schemes sort the neighbours by the four algorithms *MR*, *MF*, *MM*, and *PSMM*, respectively. To compare the strength of these routing criterions. In each run of the simulation, for each routing scheme, we pick the top-10 ranked candidates among all the nodes traversed by the task routing paths based on the routing criterion used by the scheme, which are the "best" workers with respect to the scheme and tend to be at the end of those task routing paths. Similar to the centralized case, we study the spatial closeness (the *DFO* metric) and temporal closeness from the test check-ins of the chosen workers to the simulated task.

Figure 8 compares the average *DFO* value of our scheme with those of baseline schemes over 5,000 runs of the simulation, with the maximum routing path length $TTL = 5, 10, 20$. The results demonstrate that the spatial closeness of top-ranked

workers chosen by our scheme matches the optimal solution significantly better than the baseline schemes. This shows that the routing criterion defined by our scheme can effectively guide the task routing paths to workers who are geographically close to the target venue. As for temporal closeness, Figure 9 shows our simulation results, which illustrates that our scheme can route tasks to nodes whose test check-ins are much temporally closer to the pending tasks, compared with the baseline schemes that do not take into account neighbours' temporal activeness when deciding the next hop.

V. Related Work

Previous studies have investigated various aspects of the web-based crowdsourcing platforms, including the demographics and usage behaviors of MTurk workers [11], [21], task pricing strategies [8], task search [5] and task recommendation [28] on MTurk, the relationship between financial incentives and task performance [14], and the influence of intrinsic and extrinsic motivation on task performance [20]. As the first view into user behaviors in mobile crowdsourcing markets, Musthag and Ganesan's work reveals that a small core group of workers account for a large proportion of activities generated in these systems [15].

The application of crowdsourcing has also been extensively studied by researchers. For crowdsourcing on smartphones, incentive mechanisms has been designed for mobile phone sensing [27], [10]. Crowdsourcing has also been used to defend against sybil attacks [25]. Additionally, researchers have designed algorithms for crowdsourcing to solve the human-assisted graph search [17] and maximum retrieving [24] problems.

The success of LBSNs has evoked great interest from researchers. Previous studies have investigated different characteristics of LBSNs. This includes the "Levy Flight" mobility pattern of LBSN users [4], [6], socio-spatial properties of LBSNs [18], location-focused communities in the social graph of LBSNs [3], user behaviors [12] and geo-social correlations among LBSN users [29].

VI. Conclusion

In this paper, inspired by our survey study conducted on Foursquare and MTurk, we present the centralized and decentralized design of MobiCrowd, a mobile crowdsourcing service built over LBSNs. Evaluation results demonstrate that MobiCrowd can effectively find workers who are more likely to work on mobile crowdsourcing tasks associated with different target venues by analyzing their check-in histories.

Acknowledgement

This work was supported in part by US NSF CNS-1320453, Microsoft Research Asia, CCF-NSFOCUS "Kunpeng" Research Fund, Alipay Research Fund, NSFC-61425024, NSFC-61402223, two Jiangsu Province Double Innovation Talent Programs, and NSFC-61321491. F. Xu is contact author.

References

[1] About Foursquare. https://foursquare.com/about.

[2] Yahoo! Answers. http://answers.yahoo.com/.

[3] Brown, C., Nicosia, V., Scellato, S., Noulas, A., and Mascolo, C. The importance of being placefriends: Discovering location-focused online communities. In *WOSN* (2012).

[4] Cheng, Z., Caverlee, J., Lee, K., and Sui, D. Z. Exploring millions of footprints in location sharing services. In *ICWSM* (2011).

[5] Chilton, L. B., Horton, J. J., Miller, R. C., and Azenkot, S. Task search in a human computation market. In *KDD-HCOMP* (2010).

[6] Cho, E., Myers, S. A., and Leskovec, J. Friendship and mobility: User movement in location-based social networks. In *KDD* (2011).

[7] Cohen, E., and Strauss, M. J. Maintaining time-decaying stream aggregates. *Journal of Algorithms 59*, 1 (2006), 19–36.

[8] Faridani, S., Hartmann, B., and Ipeirotis, P. G. What's the right price? Pricing tasks for finishing on time. In *AAAI Workshop on Human Computation* (2011).

[9] Gao, H., Tang, J., and Liu, H. Exploring social-historical ties on location-based social networks. In *ICWSM* (2012).

[10] Guo, B., Chen, H., Yu, Z., and et al. Taskme: toward a dynamic and quality-enhanced incentive mechanism for mobile crowd sensing. *International Journal of Human-Computer Studies 102* (2017), 14–26.

[11] Ipeirotis, P. G. Analyzing the amazon mechanical turk marketplace. *XRDS 17* (2010), 16–21.

[12] Lindqvist, J., Cranshaw, J., Wiese, J., Hong, J., and Zimmerman, J. I'm the mayor of my house: Examining why people use foursquare - a social-driven location sharing application. In *CHI* (2011).

[13] Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing 22* (2007), 395–416.

[14] Mason, W., and Watts, D. J. Financial incentives and the Performance of Crowds. In *KDD-HCOMP* (2009).

[15] Musthag, M., and Ganesan, D. Labor dynamics in a mobile micro-task market. In *CHI* (2013).

[16] Niu, W., and Kay, J. Location conflict resolution with an ontology. In *Pervasive* (2008).

[17] Parameswaran, A., Sarma, A. D., Garcia-Molina, H., Polyzotis, N., and Widom, J. Human-assisted graph search: It's okay to ask questions. In *VLDB* (2011).

[18] Pelechrinis, K., and Krishnamurthy, P. Socio-spatial affiliation networks. *Computer Communications 73* (2016), 251–262.

[19] Pickard, G., Pan, W., Rahwan, I., Cebrian, M., Crane, R., Madan, A., and Pentland, A. Time-critical social mobilization. *Science 334*, 6055 (2011), 509–512.

[20] Rogstadius, J., Kostakos, V., Kittur, A., Smus, B., Laredo, J., and Vukovic, M. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. In *ICWSM* (2011).

[21] Ross, J., Irani, L., Silberman, M. S., Zaldivar, A., and Tomlinson, B. Who are the crowdworkers? Shifting demographics in Mechanical Turk. In *CHI* (2010).

[22] Shi, J., and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 888–905.

[23] Song, L., Kotz, D., Jain, R., and He, X. Evaluating location predictors with extensive wi-fi mobility data. In *INFOCOM* (2004).

[24] Venetis, P., Garcia-Molina, H., Huang, K., and Polyzotis, N. Max algorithms in crowdsourcing environments. In *WWW* (2012).

[25] Wang, G., Mohanlal, M., Wilson, C., Wang, X., Metzger, M., Zheng, H., and Zhao, B. Y. Serf and turf: Crowdturfing for fun and profit. In *NDSS* (2012).

[26] Wang, S., and Siskind, J. M. Image segmentation with ratio cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25*, 6 (2003), 675–690.

[27] Yang, D., Xue, G., Fang, X., and Tang, J. Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing. In *MOBICOM* (2012).

[28] Yuen, M., King, I., and Leung, K. Task recommendation in crowdsourcing systems. In *CrowdKDD* (2012).

[29] Zhang, C., Shou, L., Chen, K., Chen, G., and Bei, Y. Evaluating geo-social influence in location-based social networks. In *CIKM* (2012).