Cost Minimization and Social Fairness for Spatial Crowdsourcing Tasks

Qing Liu[1], Talel Abdessalem[2], Huayu Wu[3], Zihong Yuan[3], Stéphane Bressan[1]

[1]School of Computing, National University of Singapore, Singapore
liuqing@u.nus.edu, steph@nus.edu.sg
[2]LTCI/IPAL CNRS, Télécom ParisTech, Université Paris-Saclay, France
talel.abdessalem@telecom-paristech.fr
[3] Institute for Infocomm Research, A*STAR, Singapore
{huwu,yuanzh}@i2r.a-star.edu.sg

Abstract Spatial crowdsourcing is an activity consisting in outsourcing spatial tasks to a community of online, yet on-ground and mobile, workers. A spatial task is characterized by the requirement that workers must move from their current location to a specified location to accomplish the task. We study the assignment of spatial tasks to workers. A sequence of sets of spatial tasks is assigned to workers as they arrive. We want to minimize the cost incurred by the movement of the workers to perform the tasks. In the meanwhile, we are seeking solutions that are socially fair. We discuss the competitiveness in terms of competitive ratio and social fairness of the Work Function Algorithm, the Greedy Algorithm, and the Randomized versions of the Greedy Algorithm to solve this problem. These online algorithms are memory-less and are either inefficient or unfair. In this paper, we devise two Distribution Aware Algorithms that utilize the distribution information of the tasks and that assign tasks to workers on the basis of the learned distribution. With realistic and synthetic datasets, we empirically and comparatively evaluate the performance of the three baseline and two Distribution Aware Algorithms.

Keywords: Spatial Crowdsourcing, Task Assignment, Cost, Social Fairness

1 Introduction

TaskRabbit[1] is one of a a growing number of new crowdsourcing platforms where users can outsource various tasks to crowd workers in the physical world. Some of these task require, indeed, that a worker moves to a specified location.

This activity is referred to as spatial crowdsourcing. It consists in outsourcing spatial tasks to a community of online, yet on-ground and mobile, workers. A spatial task is characterized by the requirement that workers must move from their current location to a specified location to accomplish the task.

[1] www.taskrabbit.com

For instance, in the aftermath of natural disasters (e.g., earthquakes, virus outbreaks), new platforms such as Ushahidi[2] and inSTEDD[3] help request and orchestrate actions of volunteering members of the public and independent relief forces to gather information or provide assistance. TaskRabbits wait-for-delivery service[4] finds someone to wait for a delivery and sign for a package. The assigned worker moves to the mailing address to receive and sign for the package and replace the consignee. The service alleviates the costly delivery failure problem.

Workers of the spatial crowdsourcing platform complained that the time spent on commuting between different locations to perform different tasks is non-negligible and is unpaid[1]. Therefore, in this paper, we study the assignment of spatial tasks to workers with the objective of minimizing the "commuting cost" which can refer to time and transportation fee. The spatial task is known by its location and assignment time interval and needs to be assigned to workers as it arrives. Workers have an initial position and move to the locations of the tasks to which they have been assigned. We want to find an assignment policy that minimizes the commuting cost of the workers to reach the assigned tasks. Cost refers to the commuting cost in this paper. In addition to cost optimization, we are seeking solutions that are socially fair. Namely, we are looking for solutions that minimize the variance of the workload among workers. We want to assign a similar number of tasks with a similar total cost to each worker. No worker should, preferably, be overloaded or starved.

In many applications, the spatial tasks are not randomly distributed. The distribution follows the density distribution of requesters or resources, and it may evolve over time. For example, we observed that the delivery failures occur densely in residential areas during office hours, and sparsely in those areas in the evening when most people are back home. By considering the spatial temporal distribution of tasks that can be learned from the history, we can design better strategies for the assignment.

In this paper, we study the opportunity to leverage the clustering of tasks (e.g. calls for wait-for-delivery are clustered in residential areas; calls for assistance are clustered in the disaster area). While the actual mechanisms to learn the distribution of tasks and clusters of tasks are orthogonal to the main issue discussed here, we devise algorithms that assign tasks to workers on the basis of the distribution of tasks and its evolution. Our hypothesis is that knowledge of the distributions of tasks and of its evolution not only help minimize the cost but also provide a basis for the fair assignment of tasks and cost among the workers. We evaluate the efficiency, in terms of cost minimization, and effectiveness in terms of social fairness of our proposed algorithms with several datasets. We use a realistic dataset where the schedule of spatial tasks is constructed from the log of a real parcel delivery service for an application-level evaluation. We create three synthetic datasets for the micro evaluation of the various behaviours of the algorithms. We empirically and comparatively evaluate the cost for each

[2] www.ushahidi.com
[3] instedd.org
[4] www.taskrabbit.com/m/shopping-delivery/wait-for-delivery

algorithm with each dataset. We compare the algorithms from the point of view of social fairness.

The remainder of this paper is organized as follows. Section 2 synthesizes related works. Section 3 gives a formal definition of the assignment and cost minimization problem. In Section 4, we presents the algorithms we propose. Thereafter, section 5 presents the experimental results. Section 6 concludes the paper.

2 Related Work

Several variants of the problem of assigning spatial tasks to mobile workers have been been studied under various sets of hypothesis [11,7,12,5,10,16].

The authors of [11] devise a taxonomy of spatial crowdsourcing. In their taxonomy and in its vocabulary the problem that we consider is referred to as a single task server assignment. We consider that workers that are willing to accomplish the given tasks are identified. Therefore incentives and rewards are orthogonal issues. The crowdsourcing service does not publish the spatial tasks to the workers. Instead, any online worker sends his location to the crowdsourcing server. The crowdsourcing server assign tasks to the workers. In [11], Kazemi and Shahabi consider workers constrained by their location and workload: workers can only be assigned to tasks in their neighbourhood and they can be only be assigned up to a maximum number of tasks. Given a sequence of sets tasks, the authors define and propose approximate solutions to the problem of assigning tasks to workers while maximizing the number of tasks assigned.

In [7], workers can select the tasks they want to accomplish. Given a sequence of sets tasks and workers choices, the authors define and propose approximate solutions to the problem to the problem of assigning tasks to workers while maximizing the number of tasks assigned. In [12], Kazemi et al. stressed the the validity of the results provided by the crowd workers. They studied the problem of maximizing the number of assigned tasks while ensuring that the quality of the answers reaches a confidence level. In [5], Chen et al. consider the assignment of moving workers to spatio-temporal tasks. Tasks can only be accomplished within a specified area and time interval. The authors consider moving workers who have known position, direction and speed, as well as confidence scores quantifying how reliably workers can accomplish assigned tasks. Diversity is obtained by assigning workers coming from different locations. The authors propose approximate solutions to the problem of assigning a set of workers to a set of tasks while maximizing reliability and diversity. The authors of [10,16] propose a differential privacy model for protecting location privacy of workers participating in spatial crowdsourcing tasks, which is not the concern in our problem setting.

Unlike previous works, our proposal stresses both the cost incurred by the movement of worker and the fairness of the assignment among the workers. Our objective is to minimize the total cost after completing all the tasks and the variance of the number of task and total cost per worker.

3 Problem Formulation

3.1 Preliminaries

We consider finite metric two dimensional space $\mathcal{M}$, which contains finite number of locations, on which tasks are requested. For simplicity, we will name *spatial task* as *task* in the following of this paper. A task r_j is represented as a tuple $< r_j, l_j, b_j, e_j >$ where l_j is the location where r_j is requested, and $[b_j, e_j]$ is its assignment time interval. For simplicity, we assume that there is no difference between the tasks in terms of execution time (i.e., they all need the same amount of time for their execution). For a given time step t, we denote by $R^t = \{r_1^t, r_2^t, ..., r_{n^t}^t\}$ the set of spatial tasks that can be assigned at that step ($b_j < t < e_j$), n^t denotes the total number of tasks in time step t, and n the overall number of tasks.

A worker w_i is represented as a tuple $< w_i, l(w_i), v >$ where $l(w_i)$ is the location of the worker, and v its availability. Let $W = \{w_1, w_2, ..., w_m\}$ be a set of workers who are committed to perform tasks, m be the total number of workers.

The *assignment instance* is the fact that a batch of tasks $R^t = \{r_1^t, r_2^t, ..., r_{n^t}^t\}$ is assigned to a set of workers by the platform at time step t.

Worker w_i is required to move to the locations of the tasks assigned to her (i.e., $l(w_i)^{t+1} = l_j^t$). Considering geographical constraints, a worker cannot be assigned two different spatial tasks located at different locations in the same assignment instance. A batch of n^t tasks at time step t has to be assigned to n^t different workers. In our model, each task is assigned to exactly one worker since every copy of the task can be regarded as an independent task.

We assume that the platform does not know when and where the next batch of tasks is going to arrive. Thus, it has to make instant decisions to assign this batch of tasks before the coming of the next ones. However, the platform could have some knowledge about the distribution of the whole sequence of tasks, and this distribution (i.e., $p^t : \mathcal{M} \rightarrow [0, 1]$) could be estimated by observing the data logs (history of the assignments).

Definition 1. *(Cluster). We define a cluster as a set of locations in a circle region of the space $\mathcal{M}$, where tasks are frequently requested. Each cluster (i.e., CL_i) is characterized by a weight (i.e., α_i), a center (i.e., μ_i) and the radius of the circle (i.e., $radius_i$).*

Tasks are distributed in forms of clusters, which is often the case in real world applications. The weight of a cluster is proportional to how many tasks were observed inside the cluster.

Moreover, distributions are different for different time period according to some time slots (morning, lunch time, afternoon, etc.) or to the week days, i.e., $p^{t+1} \neq p^t$ for some t.

3.2 The Assignment Cost Minimization Problem

Our spatial task assignment problem consists in moving around the m workers to perform the tasks that arise over time at some locations of the metric space $\mathcal{M}$. Tasks arrived at time step t must be treated at this step, and the assigned workers have to move to the corresponding locations to perform their tasks.

Let $c(l_1, l_2)$ denote the cost of moving from location l_1 to l_2. At each time step $t \in \mathbb{Z}^+$, n^t tasks $\{r_1^t, r_2^t, ..., r_{n^t}^t\}$ are requested at some locations $\{l_1^t, l_2^t, ..., l_{n^t}^t\}$, and each task is assigned to exactly one of the m registered workers. Let $a^t = (a_1^t, a_2^t, ..., a_{n^t}^t)$ be the assignment at time step t, each a_j^t represents the worker assigned to task r_j^t. Then, the cost at time step t is $\sum_{j=1}^{n^t} c(l(a_j^t), l_j^t)$. The unassigned workers stay their locations unchanged, i.e. $l(w_i)^{t+1} = l(w_i)^t$ for all $w_i \notin a^t$.

The goal is to determine the assignment strategy a^t at each time step t such that the cost is minimized after the completion of all the tasks, i.e. to minimize

$$\sum_{t=1}^{\tau} \sum_{j=1}^{n^t} c(l(a_j^t)^t, l_j^t) \tag{1}$$

where τ is the total number of time steps.

4 Algorithms

Our online optimization problem is a Metrical Task System[4] and is a slight variation of the k-server problem [6]. While the k-server problem considers the assignment of single workers to a schedule of individual tasks, we consider that tasks arrive set-at-a-time. Several classic algorithms[13],[14],[9],[3],[2] exist for the k-server problem that can be adapted. Noticeably we should mention the Greedy Algorithm and the Work Function Algorithm. The Greedy Algorithm simply assigns a task to the nearest worker. The Work Function Algorithm assigns tasks to workers according to the work function[13]. The work function balances between the distance of a set of worker to the set of tasks and the hypothetic distance of the workers to the tasks if an optimal algorithm had been used. The Work Function Algorithm has competitive ratio $2 \cdot \binom{|\mathcal{M}|}{m} - 1$, where $|\mathcal{M}|$ is the number of locations in the finite metric space $\mathcal{M}$. Time complexity of the Work Function Algorithm is at least $O(\binom{m}{n^t} mn^2)$, which makes it perform rather poorly in practice. The Greedy Algorithm has an unbounded competitive ratio, but it performs very well in practice with low costs. Randomized versions of the Greedy Algorithm[14], with their statistical bounds $O(m2^m)$ on the competitive ratio[9], do not yield equally good results on practical problems. Time complexity of both the Greedy and the Randomized versions of the Greedy Algorithm is $O(nm)$.

The typical situation that causes the unbounded competitive ratio for the Greedy Algorithm in the k-server problem is that where there are two workers and two tasks on a one dimensional space, a line. The workers are on the left

and the tasks on the right. The Greedy Algorithm sends the nearest worker to the first task arriving, then it sends the same worker to the second task arriving. If the schedule of task alternates between tasks at the same two positions, the first worker moves between the two tasks frantically while the second one remain idle. This is not only a situation where the Greedy Algorithm is inefficient, but also a situation where Greedy is socially unfair. A simple and natural fix is to randomize the Greedy Algorithm. The Randomized Greedy Algorithm that we consider chooses the worker to whom a task is assigned with a probability inversely proportional to the cost of the worker to the task. In the above example, randomization could eventually give a chance to the second worker to be assigned a task thus definitely unlocking the initial deadlock.

All these online algorithms are memory-less. Another design direction is, as usual with online algorithms, to add memory to the algorithm. Here we propose to devise algorithms that learn and use their knowledge of the distribution of tasks. For the sake of simplicity and because of space limitations, we do not discuss the orthogonal issue of incrementally learning the distributions. The reader can refer to the extensive literature on clustering algorithm and their incremental versions [17,8]. We consider that we are able to obtain a fairly good estimation of the distribution of tasks. Namely, we know the number of circular clusters of tasks, the coordinates of the centers, the radiuses, and the weights. This knowledge is updated as new tasks arrive. How the knowledge is updated depends on the incremental clustering or learning algorithm.

The idea that we propose in order to leverage the knowledge of the distribution is to assign workers to clusters of tasks according to the learned distribution. For each cluster, a quota of workers is calculated based on the distribution. In a first algorithm, we proactively distribute the workers to stand on the circles of the clusters. The workers are therefore on standby, waiting to be assigned a task. When a change in the distribution is observed, the workers are redeployed accordingly. However, it may not be necessary to proactively deploy the workers. In a second algorithm, we assign tasks inside a cluster to workers located outside all clusters on demand until the quota of the cluster is reached. These two algorithms can also mitigate the situation where some workers are overloaded while some others are starving, thus improve social fairness.

We present two Distribution Aware Algorithms, namely, the Proactive Distribution Aware Algorithm and the on Demand Distribution Aware Algorithm in Section 4.1.

4.1 The Distribution Aware Algorithm

The idea of the Distribution Aware Algorithm is to move a sufficient number of workers who are located outside the clusters to perform tasks requested inside the clusters (we call this movement `DEPLOY`), such that the tasks that are frequently requested inside the clusters can be served by a closer worker.

The algorithm consists of two phases: **(i)**. Estimate this "sufficient number" (i.e., quota: quo_i) of every cluster CL_i. **(ii)**. Real assignment based on the quota. Algorithm 1 gives the outline of the Distribution Aware Algorithm.

Algorithm 1: The Distribution Aware Algorithm

Input: Set of workers W, a sequence of set of tasks $R^1, ..., R^\tau$, distributions for each time period $p^1, p^2, ...$
Output: assignment $a^1, ..., a^\tau$

```
for each time period tp = 1, 2, ... do
    /* Acquire distribution information of tasks */
    CL_i: (α_i, μ_i, radius_i) (i = 1, ..., k) ← p^tp
    /* Phase (i). Estimate the quota */
    for each CL_i (i = 1, ..., k) do
        calculate quo_i for each cluster CL_i;
    /* Phase (ii). Assignment */
    Subroutine 2 (or Subroutine 3)
```

Phase (i), moving a worker located outside the clusters to perform the tasks inside the clusters incurs a heavier cost than moving a nearest worker, thus quo_i should not be too large. We set a upper bound of quo_i to be the minimum of a value proportional to the weight of CL_i and the number of locations in CL_i, minusing the existing number of workers (i.e., $exist_i$) inside CL_i. That is,

$$quo_i \leq \min\{\alpha_i \cdot m, \ \ |CL_i|\} - exist_i \tag{2}$$

$|X|$ is the cardinality of the set X (e.g., $|CL_i|$ is the number of locations inside CL_i).

On the other hand, if quo_i is insufficiently large, `DEPLOY` will not be effective, either. Here is the **benefit analysis** for the `DEPLOY`. The benefit of moving quo_i workers to CL_i is the cost saved by these quo_i workers within CL_i minus the cost of the `DEPLOY`. We want to maximize the following benefit which is the summed benefit of all clusters.

$$\sum_{Dep_1,...,Dep_k} saved\ cost\ by\ Dep_i \ - \ deploy\ cost\ of\ Dep_i; \tag{3}$$

$Dep_i \subseteq W$ $(i = 1, ..., k)$ is the set of `DEPLOY` workers to cluster CL_i, and $Dep_i \cap Dep_j = \emptyset$ if $i \neq j$. We want to determine the optimal set $\{Dep_1, ..., Dep_k\}$ that maximize Equ.3. However, the "saved cost" and the "deploy cost" is unknown since we do not know the subsequent tasks. What is more, it is inefficient to traverse all possibilities of $\{Dep_1, ..., Dep_k\}$. Thus, we calculate the benefit of the `DEPLOY` of each cluster using the following heuristic and estimation.

We sort the clusters according to their weights α_i in descending order, and estimate the benefit for each cluster as in Equ.4.

$$BNF_i = \frac{|Dep_i|}{|CL_i|} \cdot radius_i \cdot \alpha_i N^{tp} - \sum_{j=1}^{|Dep_i|} c(NN_j, \mu_i) \tag{4}$$

where $\frac{|Dep_i|}{|CL_i|}$ is the probability that a task in CL_i is served by the deployed worker. $radius_i$ is the estimated cost between two locations in the cluster. With

N^{tp} being the total number of tasks in this period, we could have the left part of BNF_i being the total estimated "saved cost by Dep_i". The right part of BNF_i is the estimated "deploy cost of Dep_i", i.e., the cost of moving the nearest $|Dep_i|$ workers (i.e., NN_j in Equ.4) located outside all clusters to the center of CL_i (i.e., μ_i). We set quo_i to be the value of $|Dep_i|$ that maximizes BNF_i under the constraint in Equ.2.

Subroutine 2: The `Proactive` Distribution Aware Algorithm

Input: Set of workers W, a sequence of set of tasks $R^1, ..., R^\tau$
Output: assignment $a^1, ..., a^\tau$

1 Sort the clusters in descending order according to the weight;
2 **for** *each cluster* CL_i $(i = 1, ..., k)$ **do**
3 Find the nearest quo_i workers to μ_i located outside all clusters;
4 Move each of the quo_i workers to the nearest point on the circle of CL_i.
5 **for** *each time step* t **do**
6 $R^t \leftarrow \{r_1^t, ..., r_{n^t}^t\}$; $W = \{w_1, w_2, ..., w_m\}$;
7 **while** $R^t \neq \emptyset$ **do**
8 $r_j^t \leftarrow$ Randomly pick a task from R^t;
9 $a_j^t \leftarrow$ The nearest worker $\in W$ to r_j^t;
10 $R^t \leftarrow R^t \backslash \{r_j^t\}$; $W \leftarrow W \backslash \{a_j^t\}$
11 Output a^t;

Phase (ii). Assignment.

The Proactive Distribution Aware Algorithm(Subroutine2). This algorithm firstly sorts the clusters in descending order according to the weight α_i (Line 1). Then, for each cluster CL_i, this algorithm proactively moves the nearest quo_i workers (nearest to the center μ_i) who are located outside all clusters to the nearest point on the circle of CL_i (Line 3,4). Workers are therefore on standby, waiting to be assigned a task. Finally, assign the tasks at one step in a random order, move the nearest available worker to perform each task (Line 8-10).

The on Demand Distribution Aware Algorithm(Subroutine3). This algorithm deploys the workers into the clusters on demand. It assigns the tasks at one step in a random order. For each task r_j^t at time step t, if it is located inside cluster CL_j^t (that has the quota quo_j^t). If $quo_j^t > 0$, move the nearest worker who is located outside all clusters to serve r_j^t (Line 6). In other cases, assign r_j^t to the nearest worker (Line 9).

In phase (i), calculating the quota for each cluster requires sorting workers with respect to cost, which takes $O(mlog(m))$. Time complexity of phase (i) is $O(kmlog(m))$ for k clusters. In phase (ii), the assignment takes $O(nm)$. Thus, Time complexity of the Distribution Aware Algorithms is $O(kmlog(m) + nm)$.

Go back to the example of two workers on a line, the Distribution Aware Algorithms will send the two workers to the two locations of the tasks based on

Subroutine 3: The `on Demand` Distribution Aware Algorithm

Input: Set of workers W, a sequence of set of tasks $R^1, ..., R^\tau$
Output: assignment $a^1, ..., a^\tau$

for *each time step* t **do**
  $R^t \leftarrow \{r_1^t, ..., r_{n^t}^t\}$; $W = \{w_1, w_2, ..., w_m\}$;
  while $R^t \neq \emptyset$ **do**
    $r_j^t \leftarrow$ Randomly pick a task from R^t;
    if $r_j^t \in CL_j^t$ *AND* $quo_j^t > 0$ **then**
      $a_j^t \leftarrow$ The nearest worker $\in W$ to r_j^t who is not in any clusters;
      $quo_j^t \leftarrow quo_j^t - 1$;
    else
      $a_j^t \leftarrow$ The nearest worker $\in W$ to r_j^t;
    $R^t \leftarrow R^t \backslash \{r_j^t\}$; $W \leftarrow W \backslash \{a_j^t\}$
  Output a^t;

the distribution. So, the cost will be zero afterwards. We do not have theoretical bounds for the Distribution Aware Algorithms.

5 Performance Evaluation

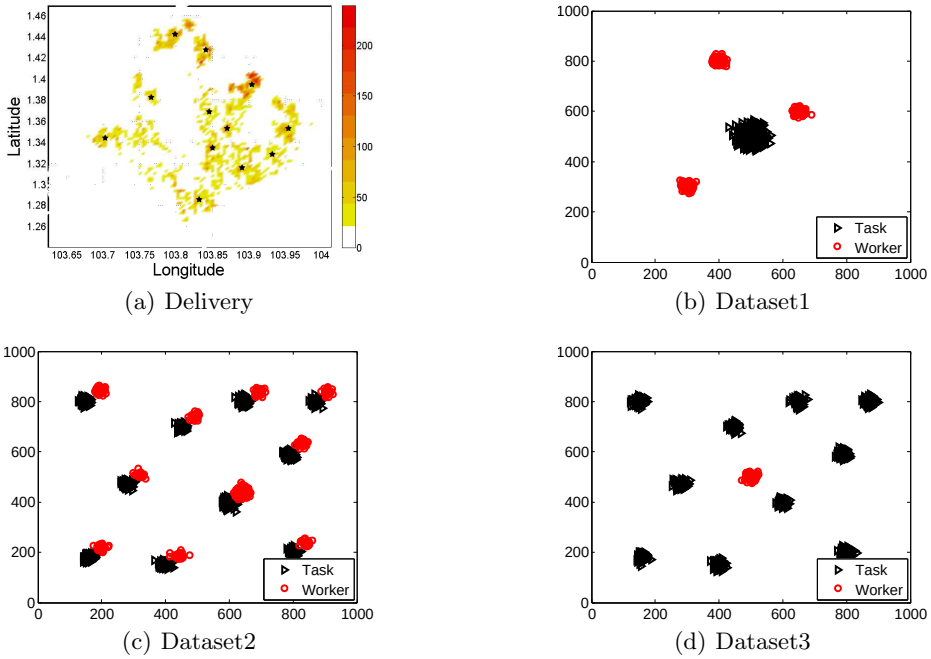
5.1 Experimental Methodology

We conduct experiments on both real world and synthetic datasets to evaluate the performance of the three baseline online algorithms, i.e., the Greedy Algorithm, the Randomized Greedy Algorithm and the Work Function Algorithm, and two Distribution Aware Algorithms, i.e., the Proactive Distribution Aware Algorithm and the on Demand Distribution Aware Algorithm. For simplicity, we will name these algorithms as `Greedy`, `Random`, `WFA`, `Proactive` and `onDemand`, respectively, in this section.

The real world dataset "Delivery" consists of records of the delivery failure packages in Singapore from April 2014 to June 2014. The distribution of those failed deliveries is presented in Fig.1(a). The color, from white to red, represents the density of the failed deliveries on the corresponding coordinates. We identify 12 clusters according to this observed distribution. The centers of the clusters are denoted by black stars in Fig.1(a). We set the radius of each cluster to be 2km and the weight of each cluster to be the density of tasks inside the cluster between April 2014 and May 2014.

We evaluate the five algorithms on the dataset of June based on the learned distributions and clusters. The number of tasks is 19k (i.e., $n = 19k$). Tasks are all in one time period. The workers are generated uniformly on the space. We vary the number of workers (i.e., m) and also the number of tasks in one time step in order to observe the behaviors of the algorithms. Euclidean distance (i.e., traveled distance) is used to measure the commuting cost between two locations for all datasets. We use the window version of `WFA`[15] (i.e., ω-`WFA`) in

the experiments considering the running time. We evaluate the `50-WFA` on the Delivery data where there is 1 task per step and $m = \{50, 100, 200\}$. In other evaluations, we only compare the performances of `Greedy`, `Random`, `Proactive` and `onDemand`.



(a) Delivery (b) Dataset1

(c) Dataset2 (d) Dataset3

Figure 1. Distributions of the Real and Synthetic Datasets

In order to evaluate the scalability and watch the behaviors of the algorithms, we generate 3 synthetic datasets named "Dataset1" (Fig.1(b)), "Dataset2" (Fig.1(c)) and "Dataset3" (Fig.1(d)) from Gaussian Mixture Distribution (i.e., GMM) on a 1000×1000 grid space. Fig.1(b)1(c)1(d) show the distributions of the tasks and the workers in the three synthetic datasets where the red circles represent the distribution of the workers and the black triangles represent the distribution of the tasks. Specifically, we use diagonal covariance matrixes for the multivariate Gaussian Distributions, and the variances for the two dimensions are set to be equal. In this way, each component in the GMM is corresponding to a cluster with the center locating at the mean, the radius equaling to the standard deviation (σ) of the x-dimension (or y-dimension). In Dataset1, 100k tasks are generated from a 1-component-GMM. All the tasks are in one time period. Workers are generated from a 3-component-GMM, the centers of which are uniformly distributed on the space. In Dataset2, there are 2 time periods. In the first time period, 50k tasks are generated from a 10-component-GMM, the means of which

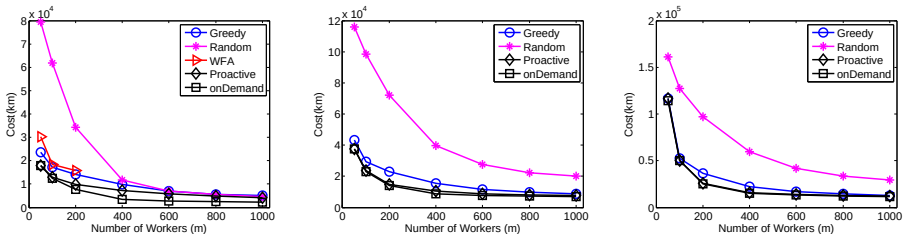
are uniformly distributed on the space. One of the components has weight 0.5, the others have weights 0.056. In the second time period, 10 components with same means as in the first period evolve all their weights to 0.1. Workers are generated from a 10-component-GMM that has different means but the same weights with the clusters of tasks in the first period. In Dataset3, there are 10 time periods. In each time period, 20k tasks are generated from a 1-component-GMM. The center of this component changes at every time period. Workers are generated from a 1-component-GMM.

Table 1. Summary of the Synthetic Datasets

	n	variance	# periods	# clusters/period	Rationale
Dataset1	100k	400	1	1	Worst Case of Greedy
Dataset2	100k	100	2	10	Evolution of Weights
Dataset3	100k	100	10	1	Moving Clusters

The summary of the datasets is in Table 1. We evaluate the performances of the algorithms by varying the number of workers and the number of tasks in one step. Each evaluation is averaged over 20 independent random cases.

5.2 Experimental Results on Cost Minimization



(a) 1 tasks/step (b) 10 tasks/step (c) 50 tasks/step

Figure 2. Results of the Delivery Data

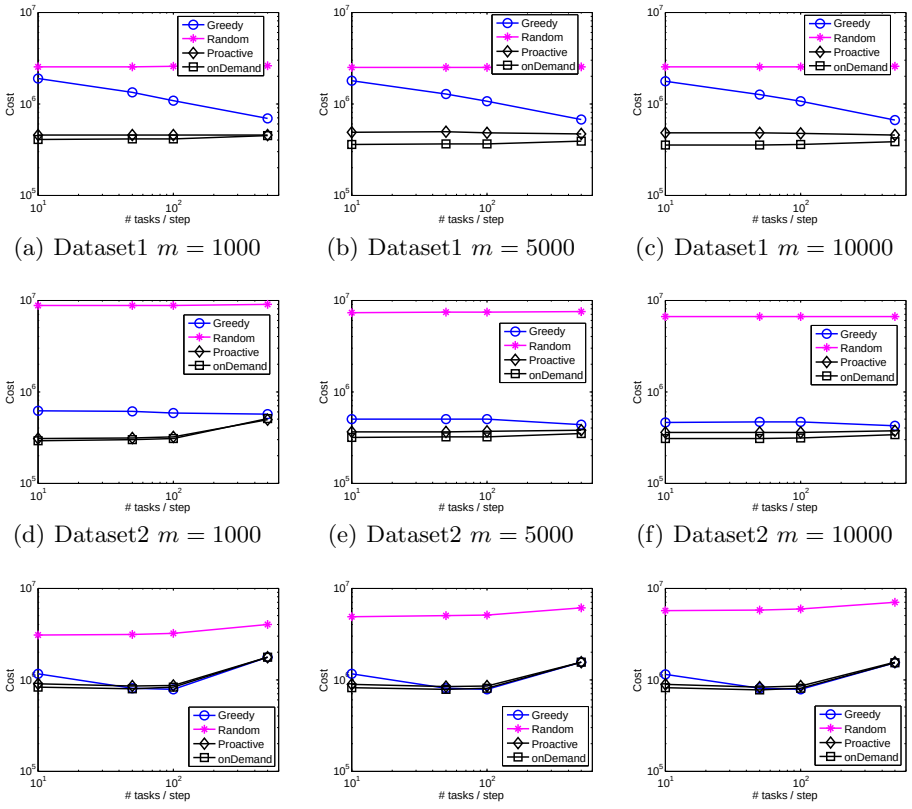
Experimental Results on Real Data. Figure 2 shows the experimental results on the Delivery dataset. Generally, as the number of worker m increases, the costs by all five algorithms decrease since there are more opportunities to assign the tasks to close workers when m is larger. `Random` performs worse than the other algorithms due to its randomness. The Distribution Aware Algorithms, `Proactive` and `onDemand`, perform better than the baseline memory-less algorithms, `onDemand` is better than `Proactive`.

Comparing Fig.2(a), 2(b) and 2(c), we can see that all algorithms perform better when the number of tasks in one step is smaller. This is because the

available worker set is larger for each task when there are fewer competitive tasks in the same time step.

When the number of workers (m) is sufficiently large, all algorithms tend to have similar performance since there are sufficient number of close workers to serve each task. As a result, the costs by all algorithms are low.

Experimental Results on Synthetic Data. Figure 3 shows the experimental results on Dateset1, Dataset2 and Dataset3. Generally, the performance of `Random` is poor, and the Distribution Aware Algorithms, `Proactive` and `onDemand`, perform better than the baseline memory-less algorithms, `onDemand` is better than `Proactive`.



(a) Dataset1 $m = 1000$ (b) Dataset1 $m = 5000$ (c) Dataset1 $m = 10000$

(d) Dataset2 $m = 1000$ (e) Dataset2 $m = 5000$ (f) Dataset2 $m = 10000$

(g) Dataset3 $m = 1000$ (h) Dataset3 $m = 5000$ (i) Dataset3 $m = 10000$

Figure 3. Results of the Synthetic Datasets

In Fig.3(a)3(b)3(c), `Greedy` performs poorly when the number of tasks in a step is small. Because `Greedy` tends to move a set of workers to the cluster and let them finish all the tasks in the cluster, which is costly.

When the number of tasks per step increases, the cost by `Greedy` decreases since the effect of moving a batch of workers in a step is similar to the effect of `DEPLOY` in the Distribution Aware Algorithms which move a set of worker outside the clusters into the clusters. This observation is contrary to the results in the Delivery dataset where we observe that costs by all algorithms increase as the number of tasks per step increases. This is because the benefit of the `DEPLOY` of `Greedy` is counteracted by a heavier cost because of the unavailability of workers in one step, which results in increasing cost in the Delivery dataset.

In Fig.3(d) and Fig.3(g), 3(h), 3(i), costs by the Distribution Aware Algorithms increase as the number of tasks in a step increases because of the unavailability of workers in one step. In this case, `Greedy` and the Distribution Aware Algorithms tend to have similar performance.

The advantage of the Distribution Aware Algorithms is that it spends a heavier cost at the beginning of each time period to deploy a number of workers to serve the tasks that are frequently requested inside clusters. This deployment benefits the following sequence of tasks. When the distributions evolves frequently as in Dataset3, the performances of Distribution Aware Algorithms are similar to `Greedy`, as shown in Fig.3(g)3(h)3(i), since the benefit of deployment is counteracted by the heavier cost at the beginning of every time period.

In conclusion, the Distribution Aware Algorithms have a lower cost of assignment compared to the baseline algorithms when the tasks are distributed in clusters.

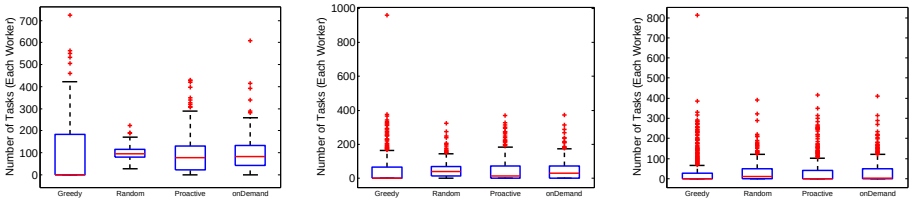
5.3 Experimental Results on Social fairness

We analyse the social fairness of the four algorithms, `Greedy`, `Random`, `Proactive` and `onDemand`, based on the Delivery data with one task per step and a number of workers varying from 200 to 600. We quantify the workload in two ways: the number of assigned tasks and the traveled distance, for every worker in the four algorithms.

Figure 4 shows the number of assigned tasks per worker by the different algorithms. We can see that `Greedy` is generally unfair, few workers are assigned a large number of tasks while most of the other workers remain idle.

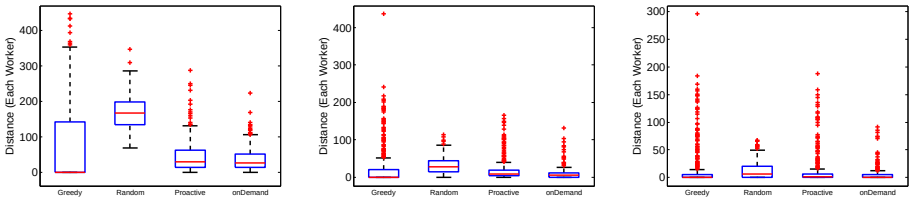
This is because `Greedy` always assigns the tasks to the nearest workers. Some workers who are initially located close to a cluster will have to finish all the tasks inside this cluster since they are always the nearest workers. This results in heavy workload of these workers.

The Distribution Aware Algorithms alleviate this problem by deploying an appropriate number of workers into each clusters. Thus, we can see from Fig.4 that `Proactive` and `onDemand` have smaller gaps between the max-workload and the min-workload than `Greedy`. `Random` is the fairest by construction. Indeed, every worker has a chance at every time step to be assigned to perform tasks.



(a) $m = 200$ (b) $m = 400$ (c) $m = 600$

Figure 4. Assigned Number of Tasks of Every Worker



(a) $m = 200$ (b) $m = 400$ (c) $m = 600$

Figure 5. Traveled Distance of Every Worker

Figure 5 shows the traveled distance per worker under the different algorithms. We can observe similar patterns as in Fig.4. Few workers traveled long distances under `Greedy`, while most of the others remain idle. `Random` is fair, however, the median cost of `Random` is higher than that of the other three algorithms. The Distribution Aware Algorithms, `Proactive` and `onDemand`, are relatively fair compared to `Greedy`, having a more balanced workload.

Clearly, fairness was built-in the design of our two Distribution Aware algorithms. It is remarkable, however, that this can be done at the benefit of a lower cost as well.

6 Conclusion

In this paper, we study the problem of assigning spatial tasks to crowd workers in the spatial crowdsourcing scenario. We formalize the assignment cost minimization problem, which is to minimize the cost of moving a set of workers to complete a sequence of spatial tasks. Then, we present three baseline and two Distribution Aware algorithms to solve this problem. We analyse the competitiveness of these five algorithms in terms of competitive ratio and social fairness. In the experiments, we compare the cost and social fairness of the five algorithms. The results show that the Distribution Aware Algorithms outperform the three baseline algorithms in terms of cost and yield a balanced workload.

References

1. http://edition.cnn.com/2015/10/22/opinions/hill-jobs-in-new-economy/.
2. N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 267–276. IEEE, 2011.
3. Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 184–193. IEEE, 1996.
4. A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)*, 39(4):745–763, 1992.
5. P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao. Reliable diversity-based spatial crowdsourcing by moving workers. *Proceedings of the VLDB Endowment*, 8(10):1022–1033, June 2015.
6. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 291–300, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.
7. D. Deng, C. Shahabi, and U. Demiryurek. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 324–333, 2013.
8. M. Ester, H. P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the VLDB Endowment*, pages 323–333, 1998.
9. E. F. Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing(STOC)*, pages 260–266. ACM, 1991.
10. L. Kazemi and C. Shahabi. A privacy-aware framework for participatory sensing. *ACM SIGKDD Explorations Newsletter*, 13(1):43–51, 2011.
11. L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 189–198. ACM, 2012.
12. L. Kazemi, C. Shahabi, and L. Chen. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 304–313, 2013.
13. E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.
14. P. Raghavan and M. Snir. *Memory versus randomization in on-line algorithms*, volume 372. Springer, 1989.
15. T. Rudec, A. Baumgartner, and R. Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, pages 1–19, 2013.
16. H. To, G. Ghinita, and C. Shahabi. A framework for protecting worker location privacy in spatial crowdsourcing. *Proceedings of the VLDB Endowment*, 7(10):919–930, 2014.
17. R. Xu. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645 – 678, 2005.