Latency-oriented Task Completion via Spatial Crowdsourcing

Yuxiang Zeng †, Yongxin Tong ‡, Lei Chen †, Zimu Zhou #
†The Hong Kong University of Science and Technology, Hong Kong SAR, China
‡BDBC, SKLSDE Lab and IRC, Beihang University, China
#ETH Zurich, Zurich, Switzerland
†{yzengal, leichen}@cse.ust.hk, ‡yxtong@buaa.edu.cn, #zimu.zhou@tik.ee.ethz.ch

***Abstract*—Spatial crowdsourcing brings in a new approach for social media and location-based services (LBS) to collect location-specific information via mobile users. For example, when a user checks in at a shop on Facebook, he will immediately receive and is asked to complete a set of tasks such as "what is the opening hour of the shop". It is non-trivial to complete a set of tasks timely and accurately via spatial crowdsourcing. Since workers in spatial crowdsourcing are often transient and limited in number, these social media platforms need to properly allocate workers within the set of tasks such that all tasks are completed (i) with high quality and (ii) with a minimal latency (estimated by the arriving index of the last recruited worker). Solutions to quality and latency control in traditional crowdsourcing are inapplicable in this problem because they either assume sufficient workers or ignore the spatiotemporal factors. In this work, we define the Latency-oriented Task Completion (LTC) problem, which trades off quality and latency (number of workers) of task completion in spatial crowdsourcing. We prove that the LTC problem is NP-hard. We first devise a minimum-cost-flow based algorithm with a constant approximation ratio for the LTC problem in the offline scenario, where all information is known a prior. Then we study the more practical online scenario of the LTC problem, where workers appear dynamically and the platform needs to arrange tasks for each worker immediately based on partial information. We design two greedy-based algorithms with competitive ratio guarantees to solve the LTC problem in the online scenario. Finally, we validate the effectiveness and efficiency of the proposed solutions through extensive evaluations on both synthetic and real-world datasets.**

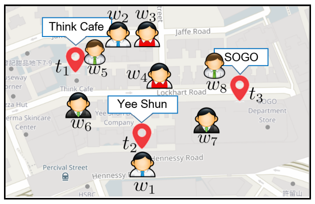
I. Introduction

With the rapid development of mobile Internet, spatial crowdsourcing emerges as a prevailing computation paradigm, where crowd workers (workers for short) are organized by the crowdsourcing platform to perform micro-tasks (tasks for short) by physically moving to the locations of the tasks. In particular, spatial crowdsourcing provides a convenient way for social media and location-based services (LBS) to acquire location-specific data by mobile Internet users. Social networks such as Facebook [1] and Foursquare [2] crowdsource information collection of POIs (*e.g.*, shops, restaurants, tourist sights) by pushing questions relevant to these locations to users who happened to check in nearby. LBS providers such as Waze [3] and OpenStreetMap [4] invite users who have just reported traffic situations or updated map data to answer additional questions about the surroundings. Naturally, the "questions" and "users" in these applications can be considered as the "tasks" and "workers" in spatial crowdsourcing.

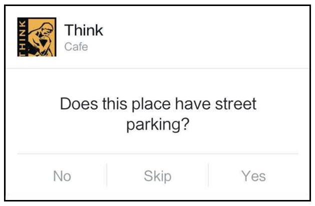
These spatial crowdsourcing powered social network and LBS applications pose new challenges in spatial crowdsourcing research. Take the Facebook Editor [5] as example, where Facebook utilizes spatial crowdsourcing to collect information about shops for Facebook maps. As shown in Fig. 1, Facebook wants to collect information about three nearby POIs, Think Cafe, Yee Shun Restaurant and SOGO Hong Kong. When one user checks in at a location near Think Cafe, Facebook immediately pushes him/her a set of questions about the three nearby POIs. Each question is a simple binary choice question such as "Does this place have street parking?". Facebook usually packs multiple questions (tasks) in a bundle and all the questions need to be completed with a desired accuracy, which will be used for queries and interactions on Facebook maps. However, it is non-trivial for Facebook to complete a set of tasks timely and accurately via spatial crowdsourcing. Since workers in spatial crowdsourcing are often transient and limited in number, these social media platforms need to properly allocate workers within the set of tasks such that all tasks are completed *(i)* with high quality and *(ii)* by a minimal amount of workers (*i.e.*, latency of task completion, estimated as the arriving index of the last worker when all tasks are completed). In fact, these social media and location-based services (LBS) platforms usually encounter the same challenges: *how to trade-off latency and quality of task completion.*

The task completion problem in the aforementioned social network and LBS applications becomes more challenging when users (workers) come dynamically (a.k.a the online scenario). In these applications, it is common that users check in on the platform at any time, *i.e.*, the platform is unaware of the arrival of users in advance. Given no information about the subsequent workers, the platform needs to immediately decide which questions (tasks) to be allocated to each new user (worker) and the decision cannot be revoked afterwards. Therefore, another challenge for task completion in spatial crowdsourcing is *how to deal with the online scenario and make arrangement among workers and tasks with partial spatiotemporal information.* To further illustrate the motivations, we go through the following toy example.

Example 1: Suppose a spatial crowdsourcing powered social media platform has three tasks t_1 - t_3 to complete (Fig. 1). Eight workers w_1 - w_8 check in at locations as shown in Fig. 1a and in an order from w_1 to w_8. Here we associate an "index" to each worker, which denotes the arriving order of the worker. Also we assume that each worker is associated with a historical accuracy for each task (Table I) and is willing to answer at most two questions from the platform.



(a) Locations of tasks and workers

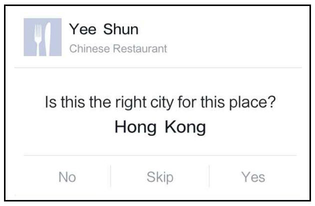


Think
Cafe

Does this place have street parking?

No | Skip | Yes

(b) Task t_1 at Think cafe

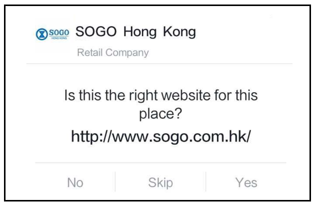


Yee Shun
Chinese Restaurant

Is this the right city for this place?
Hong Kong

No | Skip | Yes

(c) Task t_2 at Yee Shun



SOGO Hong Kong
Retail Company

Is this the right website for this place?
http://www.sogo.com.hk/

No | Skip | Yes

(d) Task t_3 at SOGO

Fig. 1: An illustration of information acquisition in a spatial crowdsourcing powered social media platform (e.g. Facebook)

TABLE I: Historical accuracy between tasks and workers

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
t_1	0.96	**0.98**	**0.98**	0.98	**0.96**	0.96	0.94	0.94
t_2	**0.98**	**0.96**	0.96	**0.98**	0.94	0.96	0.96	0.94
t_3	**0.96**	0.96	**0.96**	**0.98**	**0.94**	0.94	0.96	0.96

TABLE II: An arrangement in online scenario

	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8
t_1	**0.96**	**0.98**	**0.98**	0.98	0.96	0.96	0.94	0.94
t_2	**0.98**	**0.96**	**0.96**	**0.98**	0.94	0.96	0.96	0.94
t_3	0.96	0.96	0.96	**0.98**	**0.94**	**0.94**	**0.96**	0.96

To get high-quality answers for these three tasks, the platform often aggregates responses from multiple workers. The workers needed for each task is relevant to a quality threshold, which is preset by the platform according to different applications. However, as discussed above, workers in spatial crowdsourcing are transient and limited in number, so the platform needs to minimize the index of the last recruited worker (*i.e.*, the latency to compete all the tasks) when the quality of every task reaches the quality threshold. Here we assume the quality threshold is 2.92. For simplification, we further assume that the quality aggregation function is only the total sum of historical accuracies of the workers assigned to the task. More practical quality aggregation functions also apply (see Sec. II).

For the *offline* scenario, where the historical accuracies, locations and arriving orders of all workers are known in advance, the optimal arrangement is marked in bold in Table I and 5 workers are needed. That is, the minimal index of the last worker (latency) to complete the three tasks in the offline scenario is 5. In the *online* scenario, the historical accuracies, locations and arriving orders of all workers cannot be known beforehand. If we use the simple greedy strategy to arrange workers, one possible arrangement is shown in bold in Table II. The final latency to complete the three tasks adds up to 7, which is greater than that in the offline scenario.

As illustrated in the above example, we investigate the Latency-oriented Task Completion (LTC) problem in spatial crowdsourcing. The LTC problem aims to find a task-worker arrangement, such that the latency to complete all tasks is minimized, while the quality of the completed tasks exceeds a preset quality threshold. We also study the online version of LTC problem, which is identical to the LTC problem except that arrangements are made immediately when each new worker arrives on the platform. Note that existing research on spatial crowdsourcing either focuses on quality-based task arrangement [6] or latency-based task arrangement [7], [8], [9]. None explores the trade-off between latency and quality for task arrangement in spatial crowdsourcing. In addition, most existing research only deals with the offline scenario [8], [9], leaving them inapplicable to our LTC problem in the online scenario. In summary, we make the following contributions.

- We formulate the Latency-oriented Task Completion (LTC) problem, which aims to minimize the latency (number of workers) to complete a set of tasks with high quality in spatial crowdsourcing.
- We prove that the LTC problem is NP-hard and design a MinimumCostFlow-based algorithm to solve LTC problem in the offline scenario with a constant approximation ratio.
- We further study the LTC problem in the online scenario and design two online algorithms, Largest Acc First (LAF) and Average And Maximum (AAM), to solve the online LTC problem with constant competitive ratios.
- We verify the effectiveness and efficiency of the proposed algorithms extensively on both synthetic and real-world datasets.

In the rest of this paper, we formulate the LTC problem in Sec. II and present the solutions to the offline and the online scenarios in Sec. III and Sec. IV, respectively. We evaluate the performance of our solutions in Sec. V, review related work in Sec. VI and finally conclude in Sec. VII.

II. Problem Definition

In this section, we formally define the *Latency-oriented Task Completion* (LTC) problem. We first present the LTC problem in the offline scenario and prove it is NP-hard. Then we define the LTC problem in the online scenario, which is practical in many spatial crowdsourcing applications.

A. Basics

This subsection introduces important concepts which will be used throughout this work.

Definition 1 (Micro Task): A micro task ("task" for short) is denoted by $t =< \boldsymbol{l}_t, \epsilon >$, where $\boldsymbol{l}_t$ is the location of t and ϵ is a constant indicating the maximum tolerable error rate of t.

We assume a micro task requires a binary answer from workers, which is common in check-in based spatial crowdsourcing platforms such as Facebook Editor [5]. Without loss of generality, we denote $+1$ as a "YES" and -1 as a "NO". We make the following assumptions on tasks. *(i)* Tasks are known in advance to the platform, because spatial crowdsourcing platforms such as Facebook [5] and OpenStreetMap [4] can always prepare tasks to collect data, resolve entities and so on. *(ii)* All tasks have a constant tolerable error rate. Since spatial crowdsourcing platforms such as Facebook [5] and Waze[3] aim to help users to find the correct information about restaurants [5] or road condition [3], it is reasonable to set a single low error rate to represent the minimal guaranteed accuracy of the platform.

Definition 2 (Crowd Worker): A crowd worker ("worker" for short) $w =< o_w, \boldsymbol{l}_w, p_w, K >$, is the o_w-th person who checks in with location $\boldsymbol{l_w}$. p_w is the historical accuracy of the worker and K is a constant indicating the maximum number of tasks the worker can perform during each check-in.

We make the following assumptions on workers. *(i)* We only consider workers whose historical accuracy p_w is no less than a preset threshold ($p_w \geq 66\%$ throughout this paper). Workers whose historical accuracies are below this threshold are viewed as spams and can be reasonably ignored by the platform. *(ii)* Each worker has the same capacity, which is estimated by the platform based on surveys or human-computer interaction studies [10]. Any worker who is willing to answer more questions during each check-in can be viewed as multiple workers.

Definition 3 (Predicted Accuracy): The predicted accuracy that a worker w performs task t is measured by an accuracy function $Acc(w,t) \in [0,1]$, which takes the historical accuracy and location of worker and the location of the task as input.

In this work, we assume an accuracy function as follows.

$$Acc(w,t) = \frac{p_w}{1 + e^{-(d_{max} - \|\boldsymbol{l}_w, \boldsymbol{l}_t\|)}} \tag{1}$$

where d_{max} is the largest Euclidean distance that workers are able to perform the tasks with high accuracy. Other accuracy functions can also apply in our problem.

Definition 4 (Task Completion): Given a task t and a set of workers W_t assigned to t, the platform determines the result of t by taking a weighted majority voting among the workers. That is, $\ell_t = sign(\sum_{w \in W_t} weight_{w,t} \ell_{w,t}), \ell_{w,t} \in \{+1, -1\}$. We call a task which reaches the error rate ϵ as completed.

Note that according to the Hoeffding's inequality [11], when $weight_{w,t} = 2Acc(w,t) - 1$ and $\sum_{w \in W_t}(2Acc(w,t) - 1)^2 \geq 2\ln(1/\epsilon)$, the probability of an error in task t is less than ϵ. We define $\delta = 2\ln(1/\epsilon)$ for brevity. Based on the above definition of δ, each task should be performed at least $\lceil \delta \rceil$ times to obtain the tolerable error rate. Since this paper studies the minimal number of workers (*i.e.*, latency) when all tasks are completed, we assume that workers will accept and answer all the questions assigned to them within a short time. This assumption is reasonable because any worker who is unwilling to accept the assigned questions and answer them (*i.e.*, whose historical accept rate and answer rate are low) can be ignored automatically by the platform. We also assume that all tasks can reach the tolerable error rate.

Definition 5 (Task Latency): The latency of a task is assessed by the completion time of a task. Specifically, the latency of a task L_t is denoted as the index of the last person on the platform who performs t. Note that $L_t = \max_{w \in W_t} o_w$, if $\sum_{w \in W_t} Acc^*(w,t) \geq \delta$.

B. LTC in Offline Scenario

Based on the basic concepts above, we define the LTC problem in the offline scenario as follows.

Definition 6 (Offline LTC): Assume a set of tasks T and a tolerable error rate ϵ. Each task t has a location $\boldsymbol{l}_t$. Also assume a set of workers W, where each worker w is associated with an index o_w, a location $\boldsymbol{l}_w$, a historical accuracy p_w and capacity K. Furthermore assume an accuracy function $Acc(w,t)$ which predicts the accuracy of w performing t. The latency-orientated task completion (LTC) problem in the *offline* scenario is to find an arrangement M among tasks and workers to minimize the maximum latency of all tasks $MinMax(M) = \max_{t \in T} \max_{w \in W_t} o_w$ that meets the following constraints.

- Invariable constraint: once a task t is assigned to a worker w, the arrangement (t,w) cannot be revoked. This constraint is reasonable because the worker will see or answer the task immediately once the task is assigned to him/her.
- Capacity constraint: $\sum_{t \in T} \mathbb{1}(w \in W_t) \leq K, \forall w \in W$.
- Error rate constraint: $\sum_{w \in W_t} Acc^*(w,t) \geq \delta, \forall t \in T$ with $\delta = 2\ln(1/\epsilon), Acc^*(w,t) = (2Acc(w,t) - 1)^2$.

Theorem 1: The offline LTC problem is NP-hard.

Proof: We prove the NP-hardness of the offline LTC problem leveraging the 3-partition problem. The 3-partition problem [12] is a well-known NP-complete problem. We first reduce the 3-partition problem to the decision version of the offline LTC problem. An instance of the 3-partition problem is as follows. Given a list of $3m$ positive integers $X = \{x_1, x_2, \cdots, x_{3m}\}$ with $\sum_{i=1}^{3m} x_i = mB$ and each x_i satisfying $\frac{B}{4} < x_i < \frac{B}{2}$, the 3-partition problem aims to decide whether it is possible to partition X into m triples such that each one sums to exactly B. According to an instance of 3-partition problem, an instance of the offline LTC problem can be constructed as follows:

(1) The m triples of the 3-partition problem correspond to the m tasks with threshold $\epsilon = e^{-\frac{1}{2}}, \delta = 1$.
(2) The $3m$ elements of the list X correspond to $3m$ workers.
(3) For a worker w_i, the function $Acc^*(,)$ has the same value as $Acc^*(w_i,t) = \frac{x_i}{B} \in (\frac{1}{4}, \frac{1}{2}), \forall t \in T$, but a worker can only preform $K = 1$ task.

Given an instance of the above problem, we want to decide whether there is a feasible arrangement M such that $MinMax(M) = 3m$.

Next, we prove that an instance of the 3-partition problem is YES if and only if an instance of the offline LTC problem is YES. Since the value of $Acc^*(w,t)$ falls into the range $(\frac{1}{4}, \frac{1}{2})$, each task has to be performed by at least $\lceil \frac{\delta}{\max_{w,t} Acc^*(w,t)} \rceil > \lceil 1/\frac{1}{2} \rceil = 3$ workers. However, there are only $3m$ workers but m tasks. So each task must be performed by exactly 3 workers and each worker must perform exactly one task ($K = 1$). Otherwise at least one task cannot meet the error rate constraint. Thereby, if there is a feasible arrangement for the offline LTC problem, we will get m triples. Since a worker w performs all the tasks with the same value $Acc^*(w,.) = \frac{x_i}{B}$ and only one task is assigned to this worker, each element $\frac{x_i}{B}, x_i \in X$ appears only once in the arrangement M. Therefore, the m triples are the partition of $X' = \{\frac{x_i}{B} | x_i \in X\}$ because they are disjoint and they can cover X'. Then we need to prove that every Acc^* sum of these triples with their workers is exactly $\delta = 1$. Since the arrangement is feasible, each of the

m sums is no less than $\delta = 1$ and the sum of all these values is $\sum_{x_i \in X} \frac{x_i}{B} = \frac{\sum_{x_i \in X} x_i}{B} = m$. Thus, each of these sum is exactly 1. That is, the sum of these three x of the function Acc^* is exactly B. Hence we obtain a feasible instance of the 3-partition problem via the offline LTC problem.

From the justification above, the decision version of the offline LTC problem is NP-complete and the offline LTC problem is NP-hard. ■

C. LTC in Online Scenario

We finally define the LTC problem in the online scenario.

Definition 7 (Online LTC): Assume a set of tasks T, and a tolerable error rate ϵ, where each task t is at location $\boldsymbol{l}_t$. Also assume a set of workers W whose information is unknown before they appear, where each worker w is associated with an index o_w, a location $\boldsymbol{l}_w$, a historical accuracy p_w and a constant capacity K. Further assume an accuracy function $Acc(w,t)$ which predicts the accuracy of w performing t. The *online* LTC problem is to find an arrangement M among tasks and workers to minimize the maximum latency of all tasks $MinMax(M) = \max_{t\in T} \max_{w \in W_t} o_w$ such that

- Temporal constraint: the assignment for a new worker w must be decided immediately when s/he appears (*i.e.*, the deadline of the worker is a short time after s/he appears).
- The constraints of offline LTC are also satisfied.

Compared with the offline version, the major difference is that the platform always has no information (*e.g.*, location) about the next coming worker. Furthermore, in real applications such as Google Map [13] and Facebook [5], the assignments for each worker should be done immediately when s/he appears. This is because *(i)* the platform knows the exact place of the worker when s/he appears and his/her location is an important factor to the quality of tasks [14]; and *(ii)* workers will not stay long on spatial crowdsourcing platforms. For example, a worker may continue to drive after searching a route from Google Map [13] or have dinner after posting pictures about the restaurants to Facebook [5]. So the platform has to immediately send the worker no more than K questions about the nearby POIs. Therefore, the online LTC problem is more challenging.

As next, we first design algorithms for the offline LTC problem and then explore solutions to the online LTC problem. Note that the algorithms need to output both the arrangements among tasks and workers, as well as the maximum latency of all tasks. The maximum latency of all tasks can be measured by the last index of the worker who performs the task before the set of tasks are completed. Hence we will use the last (maximum) index of the worker in the task arrangement as the output of the algorithms in the rest of this paper.

Table III lists the notations used throughout the paper.

III. Offline Scenario

Since the offline LTC problem is NP-hard, we propose an approximate algorithm based on minimum cost flow. Compared with the optimal latency, our algorithm has a approximation ratio of 7.5. We first propose Theorem 2 to evaluate the lower bound and upper bound of the maximum latency.

TABLE III: Summary of symbol notations

Notation	Description
T, W	a set of tasks and workers
l_t, l_w	the location of task and worker
K	the capacity of each worker
ϵ, δ	the tolerable error rate of each task, the value of $2\ln(1/\epsilon)$
p_w	the historical accuracy of worker
$Acc(w,t)$	the predicted accuracy for a worker performing a task
$Acc^*(w,t)$	the value of $(2Acc(w,t)-1)^2$
Δ	the total increasement of value $Acc^*(.,.)$ for all tasks

Theorem 2 (Bounds of Maximum Latency of All Tasks): Assume $|T| \geq K$, the maximum latency of all tasks in the offline LTC problem has a lower bound of $\frac{|T|\delta}{K}$ and an upper bound of $\frac{10|T|\delta}{K} + \frac{|T|}{K} + 1$.

Proof: According to the McNaughton's Rule [15], it is easy to make an optimal arrangement when every worker has the same accuracy for each task, *i.e.*, $Acc^*(w,t) = r, \forall w \in W, t \in T$. Under this assumption, the maximum latency of all tasks is $\max\{\lceil \frac{|T|\cdot\lceil\frac{\delta}{r}\rceil}{K}\rceil,\ \lceil\frac{\delta}{r}\rceil\}$ and the optimal arrangement can be found in polynomial time. We can get the lower and upper bounds of the maximum latency of all tasks by replacing r with $\max Acc^*(,) = (2\cdot \max Acc(,)-1)^2 = (2\cdot 1-1)^2 = 1$ and $\min Acc^*(,) = (2\cdot\min Acc(,)-1)^2 = (2\cdot 0.66-1)^2 > 0.1$, respectively.

$$\begin{aligned}
bound_{lower} &\geq \max\{\lceil\frac{|T|\cdot\lceil\frac{\delta}{1}\rceil}{K}\rceil,\ \delta\}\\
&\geq \max\{\lceil\frac{|T|\delta}{K}\rceil,\ \delta\}\\
&\geq \frac{|T|\delta}{K}\\
bound_{upper} &\leq \max\{\lceil\frac{|T|\cdot\lceil\frac{\delta}{0.1}\rceil}{K}\rceil,\ \lceil\frac{\delta}{0.1}\rceil\}\\
&\leq \max\{\frac{|T|\delta}{0.1K} + \frac{|T|}{K} + 1,\ \lceil\frac{\delta}{0.1}\rceil\}\\
&\leq \frac{10|T|\delta}{K} + \frac{|T|}{K} + 1
\end{aligned}$$

■

Basic Idea. The idea of our proposed algorithm is to iteratively select a batch of workers and make an local optimal arrangement for these workers. Since in the offline LTC problem, the information about each worker is known beforehand, the optimal arrangement between a batch of workers and the uncompleted tasks can be reduced to the minimum cost flow(MCF) problem as explained shortly. Thus, the solution of the reduced MCF instance indicates the assignment for this batch of workers. Motivated by Theorem 2, we use the lower bound of the optimal result(*e.g.*, $\frac{|T|\delta}{K}$ workers) as the size of each batch to get the approximation guarantee.

Algorithm Details. We present the detailed algorithm, MCF-LTC, for the offline LTC problem as follows.

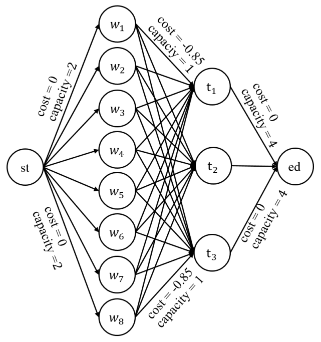
Given an instance of the offline LTC problem with workers W' in a batch, we first construct a flow network $G_F = (N_F, E_F)$, where $N_F = W' \cup T \cup \{st, ed\}$. st is a source node and ed is a sink node. For each pair $w \in W', t \in T$, there is an edge $e_F(w,t) \in E_F$ from w to t with $e_F(w,t).cost = -Acc^*(w,t)$ and $e_F(w,t).capacity = 1$. For each $w \in W'$, there is an edge $e_F(st,w) \in E_F$ from st to w with $e_F(st,w).cost = 0$ and $e_F(st,w).capacity = K$. For each $t \in T$, there is an edge $e_F(t,ed) \in E_F$ from t to ed

```
Algorithm 1: MCF-LTC
   input : $T, W, Acc^*(.,.), K, \epsilon$
   output: Maximum index of worker in a feasible
           arrangement $M$
1  $\delta \leftarrow 2\ln(1.0/\epsilon), Q \leftarrow \emptyset, m \leftarrow \frac{|T|\lceil\delta\rceil}{K}$;
2  $S \leftarrow \{0, \ldots, 0\}$ ;   /* S stores accumulated
   value for each task */
3  for $i \leftarrow 2$ to $\lceil\frac{|W|}{m}\rceil$ do
4      $W' \leftarrow$ next $\lfloor m\rfloor(\lfloor 1.5m\rfloor$ if $i = 2)$ workers from $W$;
5      construct $G_F = (N_F, E_F)$ according to $(W', T, S)$;
6      SSPA($G_F$) and construct $M'$ accordingly;
7      update the $S$ according to $M'$;
8      foreach $w \in W'$ do
9          foreach $i \leftarrow 1$ to $|T|$ do
10             if $T[i]$ has not reached $\delta$ and $w$ does not
               perform $T[i]$ according to $M'$ then
11                 push $Acc^*(w, T[i])$ with index $i$ into $Q$;
12                 maintain size of $Q$ under capacity of $w$;
13         while $Q \neq \emptyset$ do
14             extract an element with index $i$ from $Q$;
15             $S[i] \leftarrow S[i] + Acc^*(w, T[i]), M' \leftarrow$
               $M' \cup (w, T[i])$;
16     $M \leftarrow M \cup M'$;
17     if all $T$ have reached $\delta$ then break;
18 return Maximum index of worker in $M$
```

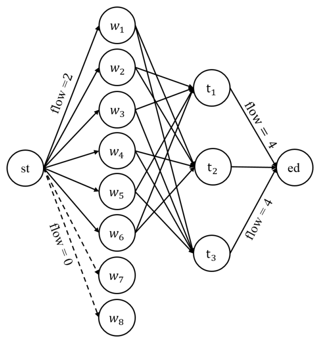
with $e_F(t, ed).cost = 0$ and $e_F(t, ed).capacity = \lceil\delta - S[t]\rceil$. Here $S[t]$ indicates the amount of Acc^* that t has already got. Note that we formulate an MCF problem, and we can get a temporary arrangement M' by solving the MCF problem. Specifically, we apply the Successive Shortest Path Algorithm (SSPA) to calculate the minimum cost flow. Similar algorithms can also apply, but SSPA is suitable for large-scale data and many-to-many matching with real-valued arc costs [16].

Note that the capacity of the edge $e_F(t, ed)$ is tightly bounded by $\lceil\delta - S[t]\rceil$. Therefore, it is possible that some workers in the same batch can still perform tasks (*i.e.*, fewer than K tasks have been assigned to the worker using M'). Hence in the next step, we allocate uncompleted tasks to workers who can still perform tasks. For these workers in the same batch, the most reliable tasks that the worker has never performed before are assigned to him/her.

Algorithm 1 shows the procedure of our MCF-LTC algorithm. In lines 1-2, we initialize the parameters. Specifically, δ is initialized based on the Hoeffding's inequality [11]. It represents the minimal accumulated Acc^* of each task. Q is a heap to maintain the largest Acc^* for each worker and is empty at the beginning. m is set as the lower bound of the maximum latency in Theorem 2, which means m workers are always enough to finish all the remaining tasks if the accuracy of each worker is 100%. We use S to store the accumulated Acc^* for each task. In lines 4-7, we first construct a flow network G_F and calculate the minimum cost flow on G_F among the workers and tasks in the current batch and then obtain a temporary matching M'. In lines 8-15, we greedily assign more tasks to the workers who can still perform tasks and update M' accordingly. Specifically, for each worker, we maintain the most reliable tasks using a heap Q in lines 9-12. Then in lines 13-15, the accumulated values of Acc^* for these tasks increase accordingly. Finally the arrangement M is updated in line 16 and the maximum index of workers in M is output as the latency to compete all the tasks.



(a) Flow graph constructed



(b) Minimum cost flow results

Fig. 2: Illustrated example of MCF-LTC

Example 2: Back to our running example in Example 1, assume the tolerable error rate is $\epsilon = 0.2$. Fig. 2a shows the flow network G_F. Based on line 1 in Algorithm 1, $\delta = 2\ln(1/0.2) = 3.22, m = \frac{|T|\lceil\delta\rceil}{K} = 6$. Since the batch size in the first round is $\lfloor 1.5m\rfloor = 9 > 8$, all the workers should be added to the flow network in the first round. Specifically, the capacity between source node st and any worker w_i is $K = 2$ and the capacity between any task t_j and sink code ed is $\lceil\delta - S[j]\rceil = \lceil 2\ln(1/0.2) - 0\rceil = 4$. Since the predicted accuracy for w_1 to perform t_1 is 0.96 (see Table I), the cost between w_1 and t_1 is $-Acc^*(w_1, t_1) = -(2Acc(w_1, t_1) - 1)^2 = -(2 \cdot 0.96 - 1)^2 \approx -0.85$. The flow graph is constructed after all the edges between w_i and t_j are added in the similar way. After running SSPA on this cost flow network, an arrangement is formed using the edges between w_i and t_j with non-zero flow, which is shown as Fig. 2b . According to this arrangement, all tasks have been completed, so there will be no updates in the next step. Therefore, the result of Algorithm 1 is 6.

Approximation Ratio. Next we study the approximation ratio (in terms of the latency to complete all tasks) of MCF-LTC. Note that a high accuracy is preferred in the real platforms like Facebook [1] and Foursquare [2]. Therefore we assume that the tolerable error rate is no more than a threshold (*i.e.*, $\epsilon \leq 0.22$) for brevity in the following analysis. This assumption is also equivalent to a preferred accuracy which is no less than 78% (*i.e.*, $1 - 0.22$).

Lemma 1: Assume $\epsilon \leq \mathrm{e}^{-1.5} \leq 0.22, \delta = 2\ln(1.0/\epsilon) \geq 3$, if the latency of the optimal arrangement is more than $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$, then the approximation ratio is 7.112.

Proof: Based on the above assumption, the optimal arrangement needs at least another worker to complete all the tasks. According to Theorem 2, our algorithm may need $\frac{10|T|\delta}{K} + \frac{|T|}{K} + 1$ workers in the worst case. Note that the number of workers required is proportional to the maximum latency to complete all tasks. Thus, the approximation ratio is

$$\frac{\frac{10|T|\delta}{K} + \frac{|T|}{K} + 1}{\lfloor\frac{1.5|T|\delta}{K}\rfloor + 1} < 6.667 + \frac{1}{1.5\delta}(1 + \frac{K}{|T|}) \leq 5 + \frac{2}{4.5} < 7.112$$

■

Lemma 2: Assume $\epsilon \le \mathrm{e}^{-1.5} \le 0.22, \delta = 2\ln(1.0/\epsilon) \ge 3$, If each task has been performed by $\lceil\delta\rceil$ workers from the first $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$ ones, Algorithm 1 will get at least $\frac{3}{4}$ of the Acc^* increase of the optimal arrangement.

Proof: Since the capacity between any task t_j and ed is no more than $\lceil\delta\rceil$, the absolute value of the cost from SSPA is no less than that of the optimal arrangement. However, this does not mean Algorithm 1 will increase Acc^* by the same amount as the optimal arrangement. This is because when $\delta = \lfloor\delta\rfloor + 10^{-9}$, Algorithm 1 might waste $1 - 10^{-9} \approx 1$ for each task in the worst case (*e.g.*, $\lceil\delta\rceil$ workers with $Acc^*(,) = 1$ perform this task). Therefore, the actual increase in Acc^* of Algorithm 1 is at least $\lfloor\delta\rfloor/\lceil\delta\rceil \ge \frac{3}{4}$ of the absolute value of the minimum cost. Thus Algorithm 1 will get at least $\frac{3}{4}$ of the Acc^* increase of the optimal arrangement. ■

Lemma 3: If the latency of optimal arrangement is no more than $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$, the sum of Acc^* in Algorithm 1 will be at least $\frac{1}{2}|T|\delta$.

Proof: Since each task in the optimal arrangement satisfies the error rate constraints, each task should have been performed by no fewer than $\lceil\delta\rceil$ workers. Otherwise the accumulation of Acc^* of at least one task is less than δ. Meanwhile, since there are $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$ workers, all these workers can perform tasks by at most $1.5|T|\lceil\delta\rceil$ times in total. Then the sum of the largest $\lceil\delta\rceil$ increase in Acc^* of each task is at least $\frac{2|T|\delta}{3}$. According to Lemma 2, Algorithm 1 would increase the accumulated Acc^* to at least $\frac{|T|\delta}{2}$ after the arrangement among the first $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$ workers. ■

Lemma 4: Assume $\epsilon \le \mathrm{e}^{-1.5} \le 0.22, \delta = 2\ln(1.0/\epsilon) \ge 3$, if the latency of the optimal arrangement is no less than $|T|\delta$ using the arrangement from the first $\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor$ workers, the approximation ratio is 7.5.

Proof: According to Lemma 3, the accumulated Acc^* in Algorithm 1 will reach at least $\frac{|T|\delta}{2}$ after arrangement among the current workers. Then according to Theorem 2, it needs at most another $\frac{\frac{1}{2}\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1$ workers. Thus, the ratio is

$$\frac{\lfloor\frac{1.5|T|\lceil\delta\rceil}{K}\rfloor + \frac{\frac{1}{2}\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1}{\frac{|T|\delta}{K}} < 6.5 + \frac{1}{\delta}(1 + \frac{K}{|T|}) < 7.5$$

■

Theorem 3: The approximation ratio of Algorithm 1 (MCF-LTC) is 7.5.

Proof: According to Lemma 1 and Lemma 4, the ratio of Algorithm 1 is $\max\{7.112,\ 7.5\} = 7.5$. ■

Complexity Analysis. Lines 4-7 take $O(\delta|T| \cdot (\frac{|T|^2\delta}{K} + (|T| + \frac{|T|\delta}{K})\log(|T| + \frac{|T|\delta}{K}))) = O(|T|^3)$ time and lines 8-15 take $O(\frac{|T|\delta}{K} \cdot |T|\log K) = O(|T|^2)$ time. The iteration takes $O(\frac{|W|K}{\delta|T|})$ batches at most. The total time cost is $O(\frac{|W|K}{\delta|T|} \cdot (|T|^3 + |T|^2) = |W||T|^2)$.

IV. Online Scenario

In this section, we design two algorithms to solve the online LTC problem with competitive ratio guarantees. In the online scenario, the decision for each new worker should be made immediately on his/her arrival, and we adopt a greedy strategy for online decision making. We first propose the Largest Acc First (LAF) algorithm, which greedily selects the tasks with the largest Acc^* for each new worker. To avoid local optimal, we further propose a hybrid greedy algorithm, Average And Max (AAM), which has a competitive ratio of 7.738.

Algorithm 2: Largest Acc First (LAF)

```
input  : $T, W, Acc^*(.,.), K, \epsilon$
output : Maximum index of worker in a feasible
         arrangement $M$
1  $\delta \leftarrow 2\ln(1.0/\epsilon), Q \leftarrow \emptyset$;
2  $S \leftarrow \{0,\ldots,0\}$ ;    /* S stores accumulated
   value for each task */
3  foreach new arrival worker $w$ do
4      foreach $i \leftarrow 1$ to $|T|$ do
5          if $T[i]$ has not reached $\delta$ then
6              push $Acc^*(w, T[i])$ with index $i$ into $Q$;
7              maintain size of $Q$ under capacity of $w$;
8      while $Q \neq \emptyset$ do
9          extract an element with index $i$ from $Q$;
10         $S[i] \leftarrow S[i] + Acc^*(w, T[i]), M \leftarrow M \cup (w, T[i])$;
11     if all $T$ have reached $\delta$ then break;
12 return Maximum index of worker in $M$
```

A. Largest Acc First (LAF) Algorithm

Largest Acc First (LAF) is a greedy algorithm for the online LTC problem. The idea is to select the tasks with the highest Acc^* for every worker. Every time a worker shows up, the tasks with the K largest Acc^* are assigned to the worker. LAF stops when all the tasks have reached δ.

Algorithm Details. When a new worker appears, the LAF algorithm enumerates all the remaining tasks and calculate the Acc^* value if the worker is to perform that task. We use a heap to maintain tasks with the K largest Acc^* for each worker. Once all the tasks reach the tolerable error rate (*i.e.*, task completed), the algorithm stops.

Algorithm 2 illustrates the LAF algorithm. In lines 4-7, the algorithm goes through all the remaining tasks to find the K tasks with the highest Acc^* for the new worker w. These K pairs of Acc^* and tasks are put into heap Q. In lines 8-10, we update the accumulated values S for each task and the arrangement M according to the selected pairs in Q. Each new worker is assigned at most K tasks on his/her arrival. Finally the algorithm outputs the maximum index of workers in the arrangement.

Example 3: Assume the same settings in Example 1. When w_1 shows up, LAF calculates $Acc^*(w_1, t) = \{(2\cdot 0.96-1)^2, (2\cdot 0.98-1)^2, (2\cdot 0.96-1)^2\} = \{0.85, 0.92, 0.85\}$. Since $K = 2$, only two tasks with the largest Acc^* (*i.e.*, t_2 and t_1) will be kept in the heap. t_2 and t_1 are assigned to w_1 and the algorithm continues to select two tasks that have not reached δ for the next worker, Similarly, t_1 and t_2 are also assigned to w_2, w_3, w_4. Then the accumulated values of Acc^* becomes $S = \{3(2\cdot 0.98-1)^2 + (2\cdot 0.96-1)^2, 2(2\cdot 0.96-1)^2 + 2(2\cdot 0.98-1)^2, 0\} = \{3.61, 3.54, 0\}$, so t_1 and t_2 satisfy the requirement of tolerable error rate. Since only t_3 has not satisfied the requirement, LAF would keep assigning t_3 to other workers. Finally, t_3 would satisfy the requirement after w_8 performs it. By running LAF, 8 workers are needed.

Competitive Ratio. Next we study the competitive ratio of the LAF algorithm.

Theorem 4: Without any assumption, the competitive ratio of any deterministic online algorithm to solve the online LTC problem is no less than 5.5.

Proof: Suppose $\delta = 1, |T| > 2, K = 1$. Further assume the first worker can perform two tasks (t_1 and t_2) equally accurate with $Acc^*(,) = 1$. Without loss of generality, t_1 is assigned to this worker. In the adversarial model, the next worker may be good at t_1 with $Acc^*(,) = 1$ but poor at t_2 with $\min_{w \in W, t \in T} Acc^*(w,t) = 0.1$. Obviously, the optimal arrangement only needs 2 workers for the two tasks (w_1 performs t_2 and w_2 performs t_1). However, no matter which task between t_1 and t_2 is assigned to w_1, the adversarial model will assign a worker who is poor at the remaining task. Accordingly, the competitive ratio is at least $\frac{1+\frac{\delta}{0.1}}{2} = 5.5$. ■

Theorem 4 shows that it is difficult to obtain the optimal arrangement when there are multiple equal values of Acc^*. Yet the theorem indicates that we can analyze the competitive ratio of LAF by calculating the proportion of Acc^* values that could have been achieved, *i.e.*, the loss of Acc^* values.

Lemma 5: Let $\lfloor\delta\rfloor = k$. LAF will get no less than $\frac{k}{2k+1}$ of the total increase of Acc^* in the optimal arrangement from the first $\lfloor\frac{|T|\lceil\delta\rceil}{K}\rfloor$ workers.

Proof: Denote the total Acc^* increase by Algorithm 2 as Δ_{LAF} and the total Acc^* increase by the optimal arrangement as Δ_{OPT}. $X = \{x_1, x_2, \cdots, x_{|T|}\}$ and $Y = \{y_1, y_2, \cdots, y_{|T|}\}$ represent the Acc^* increase of every task by Algorithm 2 and the optimal arrangement, respectively. Suppose to the contrary, $\Delta_{LAF} < \frac{k}{2k+1}\Delta_{OPT}$. It means $\frac{2k+1}{k}\sum_{i=1}^{|T|} x_i < \sum_{i=1}^{|T|} y_i$. If $\lceil\delta\rceil = k$, then every time we pick k largest values (*e.g.*, $Acc^*(,) = 1$), in the worst case we miss other k largest values as in the proof of Theorem 4. If $\lceil\delta\rceil = k+1$ (*e.g.*, $\delta = k + 10^{-9}$), then every time we pick $k+1$ largest values, only the first k values may be significant. In this case we need any worker instead of the current highly qualified worker. So in the worst case, we may miss other $k+1$ largest values after picking $k+1$ values (one is with a small Acc^* value). Accordingly, we may lose at most $\frac{k+1}{2k+1}$ of these large values. Without loss of generality, assume Algorithm 2 misses the tasks $\{t_1, t_2, \cdots, t_j\}$. Then it is obvious that

$$\sum_{i=j+1}^{|T|} x_i \geq \sum_{i=j+1}^{|T|} y_i \tag{2}$$

$$\sum_{i=1}^{j} x_i + \frac{k+1}{k}\sum_{i=j+1}^{|T|} x_i \geq \sum_{i=1}^{j} y_i \tag{3}$$

After adding these two inequations, we have

$$\sum_{i=1}^{j} x_i + \frac{2k+1}{k}\sum_{i=j+1}^{|T|} x_i \geq \sum_{i=1}^{|T|} y_i \tag{4}$$

We also know that

$$\sum_{i=1}^{|T|} y_i > \frac{2k+1}{k}\sum_{i=1}^{|T|} x_i \tag{5}$$

This contradicts with the fact that

$$\sum_{i=1}^{|T|} x_i + \frac{2k+1}{k}\sum_{i=j+1}^{|T|} x_i < \frac{2k+1}{k}\sum_{i=1}^{|T|} x_i \tag{6}$$

Hence LAF gets no less than $\frac{k}{2k+1}$ of the total Acc^* increase in the optimal arrangement from the first $\lfloor\frac{|T|\lceil\delta\rceil}{K}\rfloor$ workers. ■

Theorem 5: Assume $\epsilon \leq \mathrm{e}^{-1.5}, \delta = 2\ln(1.0/\epsilon) \geq 3$, the competitive ratio of Algorithm 2 (LAF) is 7.967.

Proof: Similar to the analysis of Theorem 3, we analyze the ratio from two sides $\Delta_{OPT} \geq \rho|T|\delta$ and $\Delta_{OPT} < \rho|T|\delta$ and infer the proper ρ to calculate the competitive ratio.

Without loss of generality, we assume $\Delta_{OPT} \geq \rho|T|\delta$ from the first $\lfloor\frac{|T|\lceil\delta\rceil}{K}\rfloor$ workers. According to Theorem 2, the optimal latency is more than $\frac{|T|\delta}{K}$. Also, LAF should have more than $\frac{3}{7}\cdot\rho|T|\delta$ increase in Acc^* based on Lemma 5($\lfloor\delta\rfloor \geq 3$). Thus according to Theorem 2, LAF needs less than $\frac{(1-\frac{3\rho}{7})\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1$ workers. Therefore the competitive ratio is

$$\begin{aligned} ratio &\leq \frac{\frac{(1-\frac{3\rho}{7})\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1}{\frac{|T|\delta}{K}} \\ &\leq 10(1-\frac{3\rho}{7}) + \frac{1}{\delta}(1+\frac{K}{|T|}) \end{aligned} \tag{7}$$

Next, according to Theorem 2, if $\Delta_{OPT} < \rho|T|\delta$, the optimal arrangement still needs at least $(1-\rho)\frac{|T|\delta}{K} + 1$ workers. Thus the competitive ratio becomes

$$\begin{aligned} ratio &\leq \frac{10\frac{|T|\delta}{K} + \frac{|T|}{K} + 1}{\lfloor\frac{|T|\lceil\delta\rceil}{K}\rfloor + (1-\rho)\frac{|T|\delta}{K} + 1} \\ &\leq \frac{10\frac{|T|\delta}{K} + \frac{|T|}{K} + 1}{(2-\rho)\frac{|T|\delta}{K}} \\ &\leq \frac{10}{2-\rho} + \frac{1}{(2-\rho)\delta}(1+\frac{K}{|T|}) \\ &\leq \frac{10}{2-\rho} + \frac{1}{\delta}(1+\frac{K}{|T|}) \end{aligned}$$

Finally, the competitive ratio is $\max\{1-\frac{3\rho}{7}, \frac{1}{2-\rho}\}\cdot 10 + \frac{1}{\delta}(1+\frac{K}{|T|})$. When $\rho = \frac{13-\sqrt{85}}{6}$, $ratio \leq 0.730\cdot 10 + \frac{2}{3} \leq 7.967$. ■

Complexity Analysis. The loop of lines 4-7 takes $O(|T|\log K)$ time and the loop of lines 8-10 takes $O(K\log K)$ time. Since the main loop of line 3 is at most $|W|$ times, the total time cost is $O(|W|(|T|\log K + K\log K)) = O(|W||T|\log K) = O(|W||T|)$.

B. Average And Max (AAM) Algorithm

The Average and Max (AAM) algorithm is a hybrid greedy algorithm inspired by the McNaughton's Rule [15]. The maximum completion time of all different processes is not only determined by the average execution time but also by the longest execution time. Similarly, the bottleneck to complete all tasks might be some difficult tasks, which need many workers to perform before the tasks can reach the tolerable error rate. Hence to complete all tasks with minimal numbers of workers, it is important to *(i)* detect and start performing difficult tasks early and *(ii)* finish other tasks with appropriate workers. AAM first uses the Largest Gain First (LGF) strategy (See Lemma 6) and switches to the Largest Remaining First (LRF) strategy when the "difficult" tasks become the bottleneck.

Algorithm Details. The key step in AAM is to maintain *(i)* the average number of workers needed to finish all tasks ("average" for short) and *(ii)* the maximum number of workers needed in any remaining task ("maximum" for short). AAM switches between the LGF and the LRF strategies based on these two values.

If the "average" is larger than the "maximum", it indicates that the total number of tasks is the bottleneck. Then AAM adopts the LGF strategy. The LGF strategy chooses the most profitable tasks for the worker. Here the most profitable task means the largest amount of increase in Acc^* without exceeding δ. Note that LGF is different from the LAF strategy proposed in Sec. IV-A. LAF selects the tasks with the largest values of Acc^* for each worker without considering the current status of the task. For example, when a task only needs a small value of Acc^* to reach δ, it is more reasonable to arrange some other tasks for this highly accurate worker. The LGF strategy is designed to account for this issue. For each task, LGF will select the highly accurate worker at the beginning (*i.e.*, when the accumulated Acc^* is much smaller than δ). When the accumulated Acc^* is close to δ, LGF will select a worker who can just cover the increase in Acc^* needed for the task. Thus the highly accurate workers are not wasted.

If the "average" is no larger than the "maximum", it indicates that some tasks may need large amounts of workers, and they are the bottleneck to complete all tasks quickly. In this case AAM adopts the LRF strategy. The LRF strategy chooses tasks that need more increase in Acc^* to reach δ so that they can be finished quickly.

Algorithm 3 illustrates the procedure. In lines 4-5, we calculate the average number avg and maximum remaining number $maxRemain$. In lines 6-12, AAM goes through all tasks to find the best K selections for the new worker w based on LGF or LRF. When the best selection is based on the highest gain(*i.e.*, LGF), which is calculated as in line 9. Specifically, the gain is the minimum between "the increase in accumulated accuracy if the task is performed by the worker (*i.e.*, $Acc^*(w, T[i])$)" and "how much accuracy is needed for this task to reach the tolerable error rate (*i.e.*, $\delta - S[i]$)". When the best selection is based on the bottleneck of the uncompleted tasks(*i.e.*, LRF), whic is calculated as in line 11. Specifically, we use "how much accuracy is still needed for this task to reach the tolerable error rate (*i.e.*, $\delta - S[i]$)" to denote the difficulty of the task. These K pairs are maintained in heap Q. In lines 13-15, the selected tasks are extracted from the heap and AAM updates the arrangement M and other state parameters. Finally, the algorithm outputs the maximum index of workers in the arrangement.

Example 4: Back to the settings in Example 1. For the first three workers, the process is the same as in LAF (see the Example 3). Now the accumulate values of Acc^* is $S = \{(2\cdot 0.96-1)^2 + 2(2\cdot 0.98-1)^2, 2(2\cdot 0.96-1)^2 + (2\cdot 0.98-1)^2, 0\} = \{2.69, 2.62, 0\}$. Thus according to line 4-5, $avg = \frac{(3.22-2.69)+(3.22-2.62)+3.22}{2} = 2.175$ and $maxRemain = \delta = 3.22$. Because $avg < maxRemain$, AAM switches to the LRF strategy. Hence both t_3 and t_2 are assigned to w_4 and t_2 would reach the requirement of error rate. Since only 2 tasks are left, t_1 and t_3 are assigned to w_5 and t_1 would reach the requirement of error rate. t_3 could reach the requirement of error rate after sending it to w_6 and w_7. Finally, the output of AAM is 7, which needs one fewer worker than LAF algorithm.

Algorithm 3: Average And Max (AAM)

```
input : $T, W, Acc^*(.,.), K, \epsilon$
output: Maximum index of worker in a feasible
        arrangement $M$
1  $\delta \leftarrow 2\ln(1.0/\epsilon), Q \leftarrow \emptyset$;
2  $S \leftarrow \{0,\dots,0\}$ ;   /* S stores accumulated
   value for each task */
3  foreach new arrival worker $w$ do
4      $avg \leftarrow \frac{\sum_{i=1}^{|T|}(\delta - S[i])}{K}$;
5      $maxRemain \leftarrow \max_{i=1}^{|T|}(\delta - S[i])$;
6      for $i \leftarrow 1$ to $|T|$ do
7          if $T[i]$ has not reached $\delta$ then
8              if $avg \geq maxRemain$ then
9                  push $\min\{Acc^*(w, T[i]), \delta - S[i]\}$ with
                   index $i$ into $Q$;
10             else
11                 push $\delta - S[i]$ with an index $i$ into $Q$;
12             maintain size of $Q$ under capacity of $w$;
13     while $Q \neq \emptyset$ do
14         extract an element with index $i$ from $Q$;
15         $S[i] \leftarrow S[i] + Acc^*(w, T[i]), M \leftarrow M \cup (w, T[i])$;
16     if all $T$ have reached $\delta$ then break;
17 return Maximum index of worker in $M$
```

Competitive Ratio. Next, we study the competitive ratio of Algorithm 3 (AAM).

Lemma 6: Algorithm 3 will always adopt the LGF strategy for the first $\frac{(|T|-K)\delta}{K}$ workers.

Proof: Suppose to the contrary, the strategy firstly switches to the LRF when the i-th worker appears, where $i < \frac{(|T|-K)\delta}{K}$. This indicates $avg < maxRemain$. Since only $i-1$ workers have come, the minimum average value can be

$$\begin{aligned} avg &\geq \frac{|T|\delta - (i-1)\cdot K \cdot 1}{K} \\ &> \frac{|T|\delta + K - \frac{(|T|-K)\delta}{K}\cdot K}{K} \\ &> \frac{K(\delta+1)}{K} = \delta + 1 \end{aligned}$$

However, $maxRemain$ must be no more than δ, which contradicts with the fact that avg should be less than $maxRemain$. It follows that AAM applies the LGF strategy for the first $\frac{(|T|-K)\delta}{K}$ workers. ■

Lemma 7: Let $\lfloor\delta\rfloor = k$. AAM will get no less than $\frac{1}{2}$ of the total Acc^* increase in the optimal arrangement from the first $\lfloor\frac{|T|\lceil\delta\rceil}{K}\rfloor$ workers.

Proof: Denote the total Acc^* increase using LGF as Δ_{LGF} and the total Acc^* increase using AAM as Δ_{AAM}.

$X = \{x_1, x_2, \cdots, x_{|T|}\}$ and $Y = \{y_1, y_2, \cdots, y_{|T|}\}$, which represent the Acc^* increase of every task using AAM and the optimal arrangement, respectively. Suppose to the contrary, $\Delta_{LGF} < \frac{1}{2}\Delta_{OPT}$. It means $2\sum_{i=1}^{|T|} x_i < \sum_{i=1}^{|T|} y_i$. Different from Lemma 5, each time LGF picks k tasks to workers, while the optimal arrangement may pick the other k tasks with the same gain. Note that we select tasks based on the gains rather than purely the accumulated accuracies (Acc^*). The tasks which have almost reached the tolerable error rate will not be assigned the highly accurate workers, which is the case with LAF. Accordingly, we may lose at most $\frac{k}{2k} = \frac{1}{2}$ of these large values. Without loss of generality, assume $\{t_1, t_2, \cdots, t_j\}$ are the tasks we miss. Then we have

$$\sum_{i=j+1}^{|T|} x_i \geq \sum_{i=j+1}^{|T|} y_i \tag{8}$$

$$\sum_{i=1}^{j} x_i + \sum_{i=j+1}^{|T|} x_i \geq \sum_{i=1}^{j} y_i \tag{9}$$

After adding these two inequations, we have

$$\sum_{i=1}^{j} x_i + 2\sum_{i=j+1}^{|T|} x_i \geq \sum_{i=1}^{|T|} y_i. \tag{10}$$

Also

$$\sum_{i=1}^{|T|} y_i > 2\sum_{i=1}^{|T|} x_i. \tag{11}$$

This contradicts with the fact that

$$\sum_{i=1}^{|T|} x_i + 2\sum_{i=j+1}^{|T|} x_i < 2\sum_{i=1}^{|T|} x_i \tag{12}$$

If follows that $\Delta_{LGF} \geq \frac{1}{2}\Delta_{OPT}$ According to Lemma 6, $\Delta_{LGF} = \Delta_{AAM}$ for the first $\lfloor \frac{|T|\lceil\delta\rceil}{K} \rfloor$ workers. Thus $\Delta_{AAM} \geq \frac{\Delta_{OPT}}{2}$ for the first $\lfloor \frac{|T|\lceil\delta\rceil}{K} \rfloor$ workers. ■

Theorem 6: Assume $\epsilon \leq \mathrm{e}^{-1.5}, \delta = 2\ln(1.0/\epsilon) \geq 3$, the competitive ratio of Algorithm 3 (AAM) is 7.738.

Proof: Similar to Theorem 5, when the total Acc^* increase of the optimal arrangement is more than $\rho n \delta$, AAM will need less than $\frac{(1-\frac{\rho}{2})\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1$ workers according to Theorem 2 and Lemma 7. In this condition, the competitive ratio is

$$\frac{\frac{(1-\frac{\rho}{2})\cdot 10|T|\delta}{K} + \frac{|T|}{K} + 1}{\frac{|T|\delta}{K}} < 10(1 - \frac{\rho}{2}) + \frac{1}{\delta}(1 + \frac{K}{|T|})$$

Otherwise the optimal arrangement would need at least another $\frac{(1-\rho)|T|\delta}{K} + 1$ workers. Thus, the competitive ratio is

$$\frac{\frac{10|T|\delta}{K} + \frac{|T|}{K} + 1}{\lfloor \frac{|T|\lceil\delta\rceil}{K} \rfloor + \frac{(1-\rho)|T|\delta}{K} + 1} \leq \frac{10}{2-\rho} + \frac{1}{\delta}(1 + \frac{K}{|T|})$$

Finally, when $\rho = 2 - \sqrt{2}$, the competitive ratio is $\max\{1 - \frac{\rho}{2}, \frac{1}{2-\rho}\} \cdot 10 + \frac{2}{3} \leq 7.738$. ■

Complexity Analysis. The calculation of the average value and maximum value in lines 4-5 takes $O(|T|)$ time. The loop of lines 6-12 takes $O(|T|\log K)$ time and the loop of lines 13-15 takes $O(K \log K)$ time. Since the main loop of line 3 is at most $|W|$ times, the total time cost is $O(|W|(|T| + |T|\log K + K\log K)) = O(|W||T|\log K)) = O(|W||T|)$.

TABLE IV: Synthetic dataset

Factor	Setting
$\lvert T\rvert$	1000, 2000, **3000**, 4000, 5000
$\lvert W\rvert$	**40000**
K	4, 5, **6**, 7, 8
Historical Accuracy	**Normal**: $\mu = [0.82, 0.84, \mathbf{0.86}, 0.88, 0.90], \sigma = 0.05$ Uniform: $mean = [0.82, 0.84, 0.86, 0.88, 0.90]$
ϵ	0.06, 0.10, **0.14**, 0.18, 0.22
Scalability	$\lvert T\rvert$ = 10K, 20K, 30K, 40K, 50K, 100K $\lvert W\rvert$ = 400K

TABLE V: Real datasets

Dataset	$\lvert T\rvert$	$\lvert W\rvert$	K	ϵ	Accuracy
New York	3717	227428	6	[0.06, 0.10, 0.14, 0.18, 0.22]	$\mu = 0.86$
Tokyo	9317	573703			$\sigma = 0.05$

V. Experimental Study

In this section, we evaluate the proposed algorithms on both synthetic and real-world datasets in terms of effectiveness (the maximum index of workers, *i.e.*, delay to complete all tasks) and efficiency (running time and memory footprint).

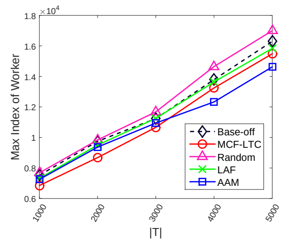
A. Experiment Setup

Datasets. We utilize a synthetic dataset to evaluate the performance in diverse settings. Table IV summarizes the settings of the synthetic dataset. The default settings are marked in bold. The locations of tasks and workers are randomly generated from a 1000×1000 2D grid. Each grid represents a 10 (meter) $\times$ 10 (meter) square. We also set $d_{max} = 30$(*i.e.*, 300 meter) for both synthetic datasets and real datasets. The setting of d_{max} comes from the real datasets which are collected by [17]. We also use the real datasets in our experiments. [17] studies the region preference (*i.e.*, the range of frequent activities around the check-in place) of users on Foursquare[2]. Therefore, according to their experiment results, a user on Foursquare is highly possible to know the POIs which are within $[10, 50]$(*i.e.*, [100(meter),500(meter)]) around his/her check-in place. Due to the limited space, we set $d_{max} = 30$ as the median of the range to avoid extreme cases.

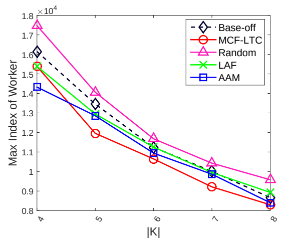
As our problem stems for commercial spatial crowdsourcing applications, we also adopt a real-world dataset for performance evaluation. Particularly, we use a dataset from Foursquare [17], which contains check-in activities of users from New York and Tokyo. We regard each user who has checks-in on Foursquare [2] as a worker. Therefore, the order of workers arrival is based on the chronological order of the check-in time of each worker. The locations of tasks in the real datasets are generated with the coordinates of POIs (*e.g.*, restaurants, shops) within the convex region of the workers accessed from Foursquare [2]. Since there is no data about the historical accuracy in the real dataset, we generate the historical accuracy of all workers to answer all nearby POIs under normal distribution. Table V presents the overview of the real-world dataset.

Baselines, metrics and implementation. We compare the performance of Base-off, Random, MCF-LTC, LAF and AAM in the evaluation. Base-off is an offline baseline where tasks with fewer workers nearby (from the remaining workers) are greedily assigned to the new worker when s/he arrives on the platform.

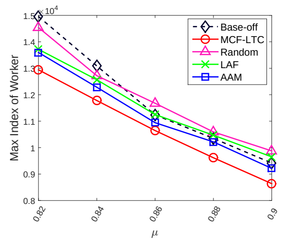
Random is a naive online baseline algorithm where tasks nearby are assigned randomly to the worker when s/he arrives on the platform. We mainly focus on the effectiveness and efficiency of each algorithm. We assess the effectiveness by



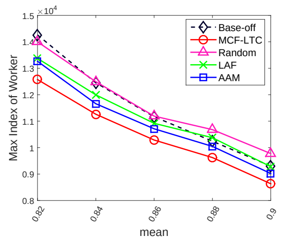
(a) Latency of varying $|T|$



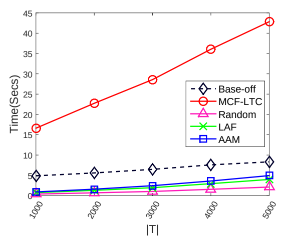
(b) Latency of varying K



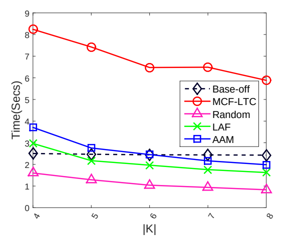
(c) Latency of accuracy(Nor.)



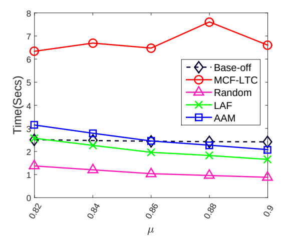
(d) Latency of accuracy(Uni.)



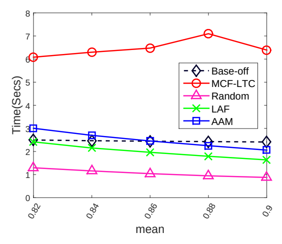
(e) Runtime of varying $|T|$



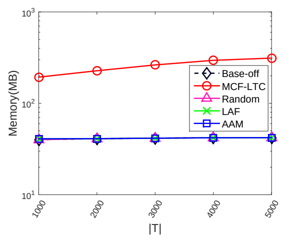
(f) Runtime of varying K



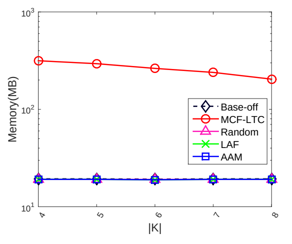
(g) Runtime of accuracy(Nor.)



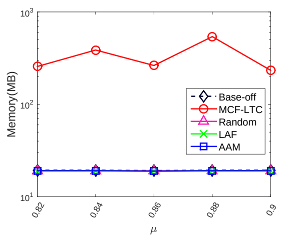
(h) Runtime of accuracy(Uni.)



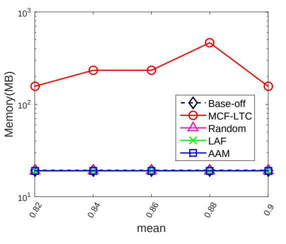
(i) Memory of varying $|T|$



(j) Memory of varying K



(k) Memory of accuracy(Nor.)



(l) Memory of accuracy(Uni.)

Fig. 3: Results on varying cardinality, capacity, and historical accuracy

the maximum index of workers, *i.e.*, the maximum latency to complete all tasks, and assess the efficiency of different algorithms via running time and memory cost. For each experimental setting, we repeat the experiment for 30 times and record the average results. All algorithms are implemented in GNU C++. The experiments are conducted on a server with 40 Intel(R) Xeon(R) E5 2.30GHz processors with Hyper-Threading enabled and 512GB memory.

B. Experiment Results

This subsection presents the performance in different settings. We first show the results on the synthetic dataset and then on the real-world dataset.

1) Effect of Cardinality T: The first column of Fig. 3 shows the results by varying the tasks T.

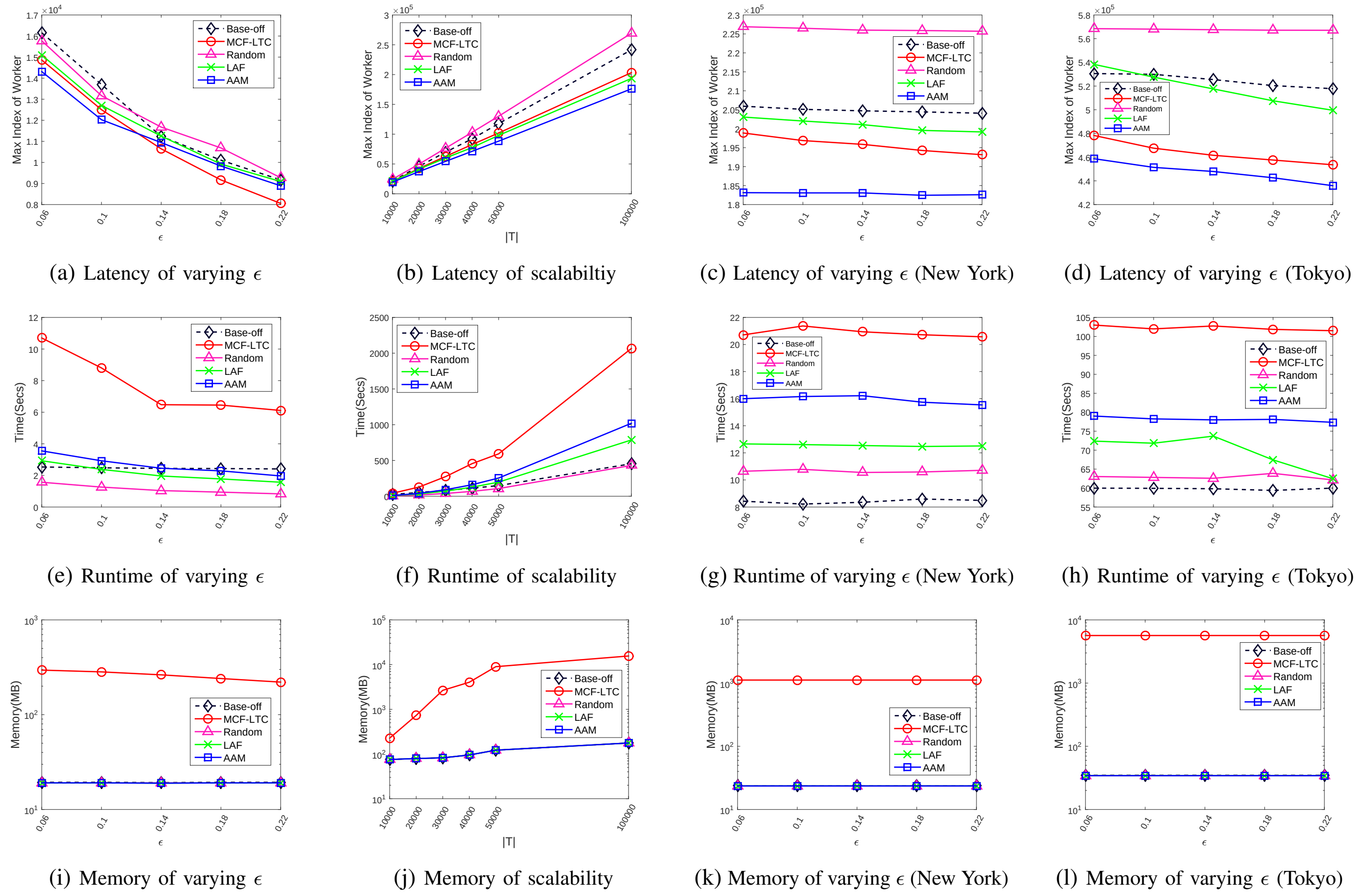
Effectiveness. Comparing the offline algorithms, MCF-LTC always achieves the smaller maximum index of workers than Base-off. Among the online algorithms, AAM performs the best and both LAF and AAM perform better than Random. For large T (*e.g.*, $|T| = 5000$), AAM performs even better than the proposed offline algorithm (*i.e.*, MCF-LTC). This is because a large T leads to a large batch. Some workers in a large batch may have a large index, but they can perform the tasks better than the others. MCF-LTC tends to select these workers with large indices, leading to larger latency. For example, when $|T| = 5000$, the size of a batch is 3333 and the outputs of both AAM and MCF-LTC fall into the same batch. But since MCF-LTC always picks the worker with the highest accuracy, it may choose the one whose index is the last in the batch.

Efficiency. Both the running time and the memory cost increase with the increase of $|T|$. MCF-LTC incurs the longest running time and the largest memory cost. AAM and LAF need more time than Random because both of them need to maintain a heap. Yet their memory costs are similar as Random because the size of the heap is no more than a small value K. AAM incurs slightly more overhead than LAF because the former needs to calculate the "average" and "maximum" values to switch between two strategies.

2) Effect of Capacity K: The second column of Fig. 3 illustrates the results by varying the capacity of workers.

Effectiveness. The maximum index of workers of all algorithms drops with the increase of K. This is because workers can perform more tasks on average while the total number of tasks is fixed. The most notable drop in the maximum index of workers is when the capacity increases from 4 to 5. It indicates a slight increase in capacity may notably speed up task completion when the capacity is small. As expected, AAM always yields smaller maximum task latency than the other two online algorithms. The offline algorithm, MCF-LTC, performs better than Base-off. Note that when capacity is small(*e.g.*, 4), AAM outperforms MCF-LTC. This can be explained by the large batch size when the capacity of workers is small.

Efficiency. In terms of running time and memory, MCF-LTC consumes the most. Both AAM and LAF consume more time than Random but with similar memory cost.


(a) Latency of varying ϵ (b) Latency of scalabiltiy (c) Latency of varying ϵ (New York) (d) Latency of varying ϵ (Tokyo)

(e) Runtime of varying ϵ (f) Runtime of scalability (g) Runtime of varying ϵ (New York) (h) Runtime of varying ϵ (Tokyo)

(i) Memory of varying ϵ (j) Memory of scalability (k) Memory of varying ϵ (New York) (l) Memory of varying ϵ (Tokyo)

Fig. 4: Results on tolerable error rate, scalability and real datasets

3) Effect of Distribution of Historical Accuracy: The third and the fourth columns of Fig. 3 present the results when the historical accuracy follows a normal distribution and a uniform distribution, respectively.

Effectiveness. For historical accuracy following a normal distribution, MCF-LTC performs better than Base-off. Both LAF and AAM outperform Random. For historical accuracy following a uniform distribution, MCF-LTC is still the best among the offline algorithms while AAM is the best among the online algorithms.

Efficiency. For historical accuracy with both distributions, MCF-LTC always consumes more time and memory than Base-off. AAM and LAF consume more time than Random, but have similar memory cost.

4) Effect of Tolerable Error Rate ϵ: The first column of Fig. 4 shows the results when varying the tolerable error rate.

Effectiveness. The maximum index of workers of all the algorithms drops with the increase of ϵ. MCF-LTC performs better than Base-off and AAM performs the best among the three online algorithms.

Efficiency. The results for running time and memory cost are similar to those in previous experiments.

5) Scalability: The results on scalability are shown in the second column of Fig. 4.

Effectiveness. When increasing the number of tasks from 10 thousand to 100 thousand, MCF-LTC always outperforms Base-off and the two proposed online algorithms perform better than Random. For very large-scale tasks, AAM outperforms the other two online methods.

Efficiency. MCF-LTC becomes inefficient with very large numbers of tasks. However, AAM and LAF are still efficient in both running time and memory cost compared to Random. Although AAM takes longer time, the gap in time is marginal compared with the other two online algorithms.

6) Performance on Real-world Dataset: The third and the fourth columns of Fig. 4 show the results.

Effectiveness. Again, MCF-LTC is the best offline algorithm while AAM is the best among the online algorithms.

Efficiency. The results for running time and memory cost are similar to those in previous experiments.

C. Experiment Summary

We make the following observations.

- AAM is the most effective online algorithm (yields the smallest maximum index of workers) especially with large amounts of tasks. In most cases, AAM outperforms Random and LAF in terms of effectiveness.
- MCF-LTC always peforms better than Base-off and the effectiveness is affected by the batch size.
- MCF-LTC incurs significant running time and memory cost. LAF is the best in terms of time and memory.
- AAM is comparatively effective, efficient and scalable among the online algorithms.

VI. Related Work

Our work is related to the following categories of research: quality control, latency control and spatial crowdsourcing.

A. Quality Control in Crowdsourcing

Quality control is an important issue in crowdsourcing research, because workers can make mistakes. There are two ways to ensure the quality of task completion. The first is *Truth Inference* [18], where one task is assigned to multiple workers and the result is inferred by aggregating and mining answers from multiple workers. The second is *Task Assignment* [19], [20], where tasks are only assigned to the workers who are more likely to give correct answers. Our work aims to control not only the quality of task completion, but also the maximum latency of task completion. Hence previous quality control approaches are inapplicable to our problem.

B. Latency Control in Crowdsourcing

For tasks that require timely answers, crowdsourcing platforms strive to minimize the latency of task completion. Most research attracts workers to complete tasks quickly via monetary incentives [7], [8]. Yet we explore latency control with voluntary workers, which is the case in many spatial crowdsourcing platforms such as Facebook Editor [5] and Google Map Marker [13]. Hence conventional latency control methods are not directly applicable to our problem. One close work is [9], where three latency control methods for crowdsourcing are proposed, including straggler mitigation, pool maintenance and hybrid learning. However, it assumes that workers are always available on the platform. In spatial crowdsourcing, the availability of workers are relatively short-lived, and spatiotemporal information poses additional constraints on tasks and workers.

C. Spatial Crowdsourcing

Recently, spatial crowdsourcing has raised increasing research interest [21]. The majority of research investigates *Quality control* [6] and *Task Assignment* [22], [23], [24], [25], [26] in spatial crowdsourcing. Yet the latency for task completion is largely overlooked in the current research. Therefore our work tries to fill in this gap. To the best of our knowledge, this is the first work to account for the trade-off between quality and latency in spatial crowdsourcing.

VII. Conclusion

In this paper, we identify the Latency-oriented Task Completion (LTC) problem in spatial crowdsourcing. We first prove the NP-hardness of the LTC problem. For the LTC problem in the offline scenario, where the information of tasks and workers is known in advance, we design a minimum-cost-flow based algorithm, called MCF-LTC, with a constant approximation ratio. We then study the online scenario of the LTC problem, where workers arrive on the platform dynamically and the platform should arrange tasks for each worker immediately on his/her arrival. We propose two efficient greedy-based online algorithms with constant competitive ratio guarantees to solve the LTC problem in the online scenario. Finally, we conduct extensive experiments to verify the effectiveness, efficiency and scalability of the proposed solutions on both synthetic and real-world datasets.

Acknowledgment

Yuxiang Zeng and Lei Chen's works are partially supported by the Hong Kong RGC GRF Project 16207617, National Grand Fundamental Research 973 Program of China under Grant 2014CB340303, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Webank Collaboration Research Project, and Microsoft Research Asia Collaborative Research Grant. Yongxin Tong's work is partially supported by National Science Foundation of China (NSFC) under Grant No. 61502021 and 61532004. Yongxin Tong and Lei Chen are the corresponding authors in this paper.

References

[1] "Facebook," *https://www.facebook.com*.

[2] "Foursquare," *https://www.foursquare.com*.

[3] "Waze," *https://www.waze.com*.

[4] "Openstreetmap," *http://www.openstreetmap.org*.

[5] "Facebook editor," *https://www.facebook.com/editor*.

[6] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: trustworthy query answering with spatial crowdsourcing," in *GIS 2013*.

[7] Y. Gao and A. Parameswaran, "Finish them!: Pricing algorithms for human computation," *PVLDB 2014*.

[8] V. Verroios, P. Lofgren, and H. Garcia-Molina, "tdp: An optimal-latency budget allocation strategy for crowdsourced maximum operations," in *SIGMOD 2015*.

[9] D. Haas, J. Wang, E. Wu, and M. J. Franklin, "Clamshell: Speeding up crowds for low-latency data labeling," *PVLDB 2015*.

[10] M. Haklay, "How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets," *Environment and planning B: Planning and design 2010*.

[11] C.-J. Ho, S. Jabbari, and J. W. Vaughan, "Adaptive task assignment for crowdsourced classification," in *ICML 2013*.

[12] R. G. Michael and S. J. David, "Computers and intractability: a guide to the theory of np-completeness," *WH Free. Co., San Fr*, 1979.

[13] "Google map marker," *https://www.google.com/mapmaker*.

[14] H. Hu, Y. Zheng, Z. Bao, G. Li, J. Feng, and R. Cheng, "Crowdsourced poi labelling: Location-aware result inference and task assignment," in *ICDE 2016*.

[15] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science, 1959*.

[16] M. L. Yiu, K. Mouratidis, N. Mamoulis *et al.*, "Capacity constrained assignment in spatial databases," in *SIGMOD 2008*.

[17] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015.

[18] Q. Li, Y. Li, J. Gao, L. Su, B. Zhao, M. Demirbas, W. Fan, and J. Han, "A confidence-aware approach for truth discovery on long-tail data," *PVLDB 2014*.

[19] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "Slade: A smart large-scale task decomposer in crowdsourcing," *IEEE Transactions on Knowledge and Data Engineering 2018*.

[20] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *SIGMOD 2015*.

[21] Y. Tong, L. Chen, and C. Shahabi, "Spatial crowdsourcing: Challenges, techniques, and applications," *PVLDB 2017*.

[22] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *GIS 2012*.

[23] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE 2016*.

[24] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: experiments and analysis," *PVLDB 2016*.

[25] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *PVLDB 2017*.

[26] Y. Zhao, Y. Li, Y. Wang, H. Su, and K. Zheng, "Destination-aware task assignment in spatial crowdsourcing," in *CIKM 2017*.