Received August 14, 2019, accepted August 29, 2019, date of publication September 9, 2019,
date of current version September 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2940028

Enhancing User Experience of Task Assignment in Spatial Crowdsourcing: A Self-Adaptive Batching Approach

LAI QIAN[1], GUANFENG LIU[2], FEI ZHU[1], ZHIXU LI[1], YU WANG[1], AND AN LIU[1]
[1]School of Computer Science and Technology, Soochow University, Suzhou, China
[2]Department of Computing, Macquarie University, Sydney, NSW, Australia

Corresponding author: Zhixu Li (zhixuli@suda.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 61572336, Grant 61572335, Grant 61632016, Grant 61772356, Grant 61802344, Grant 61602400, and Grant 61702227, in part by the Natural Science Research Project of Jiangsu Higher Education Institution under Grant 18KJA520010 and Grant 17KJA520003, in part by the Priority Academic Program Development of Jiangsu Higher Education Institutions, and in part by the Hui-Chun Chin and Tsung-Dao Lee Chinese Undergraduate Research Endowment (CURE).

ABSTRACT Faced with the explosive demand of real-world applications, spatial crowdsourcing has attracted much attention, in which task assignment algorithms take the dominant role in the past few years. On the one hand, most recent studies concentrate on maximizing the overall benefits of the platform, ignoring the fact that user experience also plays an essential role in task allocation. On the other hand, they focus on matching, that is, how to assign tasks, rather than batching, that is, when to make assignment. In fact, user experience also depends on batching, but this is largely overlooked by current studies. In this paper, we propose a self-adaptive batching mechanism to enhance user experience in spatial crowdsourcing. With appropriate start-up timestamps, previous matching methods can perform better. Multi-armed bandit algorithm in reinforcement learning is adopted to split the batch dynamically according to historical current states. Extensive experimental results on both real and synthetic datasets demonstrate the effectiveness and efficiency of the proposed approach.

INDEX TERMS Spatial crowdsourcing, online task assignment, user experience, self-adaptive batching, multi-armed bandit.

I. INTRODUCTION

With the rapid development of mobile Internet and sharing economy, a novel framework called spatial crowdsourcing (SC) is proposed to meet with the growing demand for suitable solutions to the problem of allocating spatial tasks. Nowadays, there are many typical real-world SC applications, such as car-hailing service (*e.g.*, Uber and DiDi), food delivery service (*e.g.*, Eleme and Meituan) and handyman service (*e.g.*, TaskRabbit). There are typically three parties in SC applications: workers, tasks, and a platform. Workers and tasks arrive dynamically to the platform which is in charge of matchmaking, that is, tasks are assigned to workers. A remarkable feature of SC is that workers are required to travel to corresponding locations physically to finish their tasks. Task assignment is a crucial problem in SC and has received much attention in recent years.

Most existing works on task assignment focus on improving the overall performance of the platform. For example, [1], [2], [13] try to maximize the number of assigned tasks; [8], [14], [15] intend to maximize the overall utility score of the platform; and [16], [17] aim to minimize the total travel cost of the platform. While the performance of the platform is important to successful SC applications, user experience also plays a significant role in SC. This is because users are the cornerstone of SC applications. Only by considering the interests of users, a platform can stand out from the competition with other platforms. A few studies on improving user experience have been reported very recently. In most SC applications, user waiting time is an important index of user experience. Based on this observation, [18] presents a task assignment method to minimize the maximum user waiting time.

The associate editor coordinating the review of this manuscript and approving it for publication was Shirui Pan.

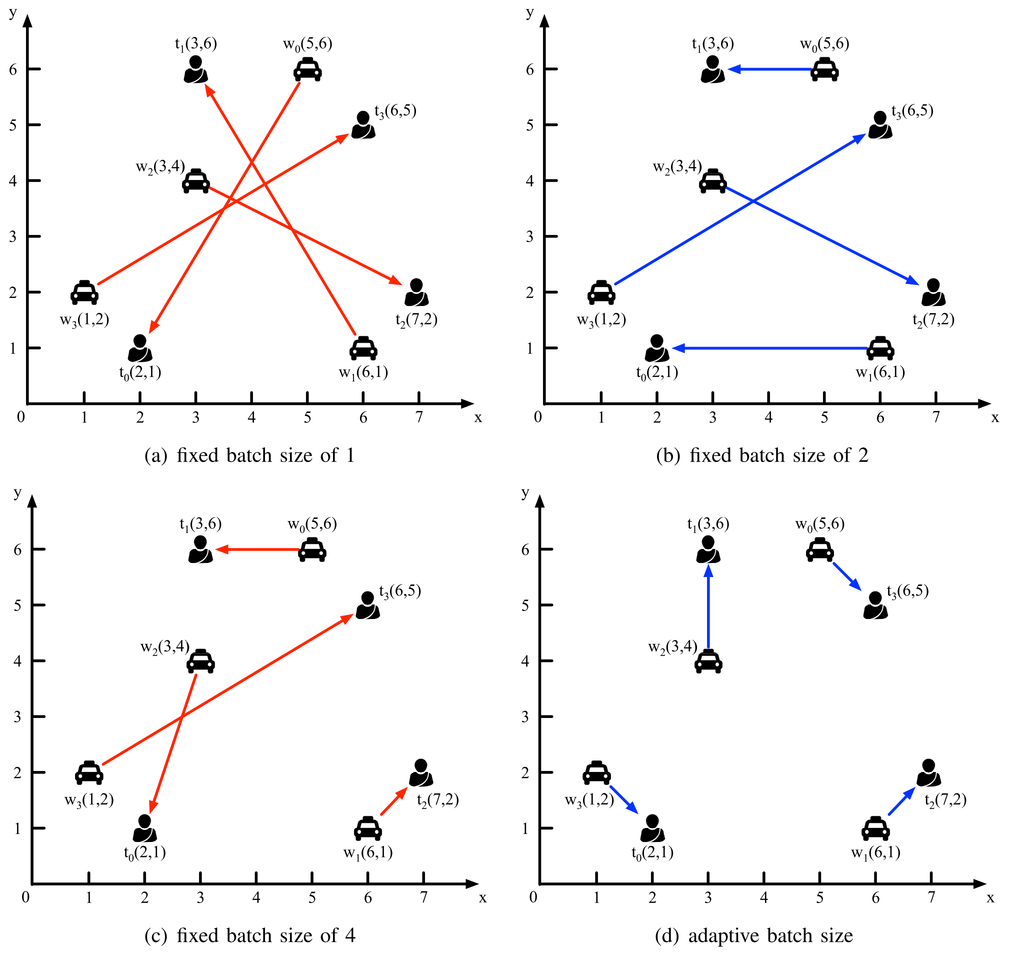


FIGURE 1. An example of 4 passengers and 4 taxis.

Though the above method can effectively alleviate the worst user waiting time, it is essentially a task assignment strategy, that is, how to assign tasks to workers. The problem of when this strategy should be executed, however, has been largely overlooked recently. Existing solutions to this problem is either real-time or fixed batch. However, both of them sometimes have unsatisfied performance in terms of user experience. We illustrate this using the following car-hailing example.

Assume a car-hailing platform has 4 car-hailing task (t_0-t_3) and 4 workers (w_0-w_3). The locations and the arriving time of tasks and workers are presented in Fig.1, respectively. The batch size here represents the time interval between batch assignments. That is, if the batch size is set to 2, the platform will wait for 2 units of time after each allocation and then perform the next allocation. In this example, we do not consider the worker waiting time, and if the worker is not assigned to the task, he will be treated as an idle worker to join the next batch for further allocations. Firstly, we set the batch size to 1, which equate to the online assignment because incoming tasks are immediately assigned to available workers within the unit time of its arrival. Under this circumstance, we match w_0 with t_0 in the first place since w_0 is the only available worker when t_0 arrives at time 1. We assume that each worker's velocity is 1, so each worker's traveling distance is equivalent to his/her traveling time. If we regard the waiting time as the cost of the allocation, then the cost of this single process is $\sqrt{34}$(traveling time of w_0) + 0(delay of t_0). The solution under the batch size of 1 with t_0, t_1, t_2, t_3 matched to w_0, w_1, w_2, w_3 at time 1, 3, 5, 8 respectively are demonstrated in Fig.1(a). The total cost is $\sqrt{34} + \sqrt{34} + \sqrt{20} + \sqrt{34} \approx 21.96$, and Fig.1(a) also conveys the message that real-time allocation may lead to an unbalance distribution because users sacrifice a nearer car for shorter waiting time. In terms of fixed batch, we adjust the batch size to 2. During this process, we match t_0, t_1, t_2, t_3 with w_1, w_0, w_2, w_3 at time 2, 4, 6, 8 respectively and the total cost of which is $5 + 3 + (\sqrt{20} + 1) + \sqrt{34} \approx 19.30$. From the result above we can see a slight decrease in the total cost due to a more reasonable allocation. With a larger batch size, we can work out a local optimal solution with existing allocating methods. Furthermore, if we set the batch size to 4, the total cost is $(\sqrt{10} + 3) + 3 + (\sqrt{2} + 3) + \sqrt{34} \approx 19.40$, which is slightly larger than when the batch size is 2. This is because even if

TABLE 1. Arrival time of car-calling tasks (passengers) and workers (cars).

time	0	1	2	3	4	5	6	8
task/worker	w_0	t_0	w_1	t_1	w_2	t_2	w_3	t_3

the distribution results become more reasonable, the first ones that arrive must wait until the end of the current batch to get assigned, thus increasing the average waiting time. Therefore, we divide the current time period into two batches according to the distribution of workers and tasks. The first batch size is 6, and the second batch size is 2. The specific allocation is as shown in Fig.1(d). As we can see from 1(d), The best result is that t_2 is assigned to w_1, t_0 is assigned to w_3. As for whether t_1 is assigned to w_0 or w_2, since the two workers are the same cost to t_1, a worker will be randomly assigned to t_1. In our current example, we assign t_1 to w_2, so this is the case where w_0 is idle and added to the second batch for allocation. Therefore, there will be a match between t_3 and w_0 in the second batch. We can have an ideal allocation whose total cost is $(\sqrt{2} + 5) + 5 + (\sqrt{2} + 1) + \sqrt{2} \approx 15.24$, about 20% less average waiting time than before, indicating that the problem of when to execute the allocation can play a significant role in optimizing user's average waiting time.

In this paper, we propose a self-adaptive batching approach in SC platform with the purpose of minimizing requesters' average waiting time. Compared with similar work [8], our work focuses on the user experience instead of the overall utility of the platform, and also we will focus on the timing of allocation and the impact of current real-time user supply and demand on batch size settings. In order to achieve our experimental goals, we need a way to dynamically adjust the batch size based on the current situation. We model the self-adaptive batching approach as the task of repeatedly selecting a batch size when a new batch begins. Therefore, we introduce the multi-armed bandit problem(MAB) into our mechanism and make targeted improvements to the ε-greedy method of the MAB based on our model, through which we can give scores for different batch sizes and decide whether to continue selecting the historical optimal batch size or to explore a batch size that better matches the current situation.

Based on the above discussion, we summarize our main contributions as follows:

- We propose to use the batching method to optimize the user experience, which is the first work to consider this problem in SC framework.
- We present a MAB-based self-adaptive batching approach to optimize SC task assignment, which can adapt to conventional assignment algorithms and the changing of real-time supply-demand relationship.
- The effectiveness and efficiency of our mechanism are veriïĄed on both synthetic and real-world datasets.

The rest of the paper is organized as follows. In Section II, we review some related works. We next demonstrate the process and related definitions to form a basic concept for our mechanism in Section III. Our solutions and its central idea will be elaborated in Section IV. Extensive evaluations on both synthetic and real-world datasets are presented in Section V. Finally, we make our conclusions in Section VI.

II. RELATED WORK

In this section, we review some typical works related to our problem from two categories, spatial crowdsourcing and online task-assignment and explain the differences between ours and the mentioned mechanisms.

Existing algorithms generally build their models on bipartite matching problem to cater for different needs. Reference [3] design a general-purpose SC platform to solve the maximal assignment problem. [4] consider recommending an online optimal path for SC worker so that they can achieve the most reward by completing tasks along the way. Reference [5] put forward a prediction-based online task assignment model to enhance the probability of successful assignment. Reference [6] designs multi-skill algorithms to maximize the revenue of workers under the constraints of budgets and skills. Reference [7] take that workers may reject the task assigned from the SC platform into consideration. Reference [8] propose a reinforcement learning approach to find a matching allocation that yields the highest total revenue.

Existing online assignment algorithms [1], [9], [16], [20] are normally classified into real-time mode and batch mode. Real time mode hardly makes sure the efficiency of assignment because better matching objects often appear after the allocation. Nowadays, most works use the batch mode as a fixed framework by setting a specific time window value, and never pay their attention on the truth that, the size of each batch also matters a lot. Reference [8] firstly use adaptive allocation by calculating and interrupting current batch dynamically and achieved remarkable results, which also proves the significance of our proposal – focusing on when platform takes action of assignment under the online scenario to better the assignment effectiveness, rather than which allocation algorithm we need to utilize.

III. PROBLEM STATEMENT

We first give some basic definitions related to our SC platform, then formally define the problem we aim to optimize and discuss the criteria we evaluate a given deterministic self-adaptive batching algorithm's performance.

A. SC PLATFORM-RELATED DEFINITIONS

Definition 1 (Worker): A worker on spatial crowdsourcing platform is denoted by $w_i = (l_i, v_i)$, where l_i and v_i respectively indicates his/her current location and unchangeable moving speed. And a location is a geographic spot $l = (longitude, latitude)$ where longitude, latitude respectively indicated the longitude and latitude value.

In this paper, when a worker becomes available, we will record his/her location at that moment. In addition, we assume an available worker will not change his/her location until he/she is assigned to a task.

Definition 2 (Task/Request): A task or a request on SC platform is denoted by $t_j = (l_j, s_j)$, where l_j and s_j respectively indicates the location where this request need to be completed and the timestamp when this request appears on the platform.

Definition 3 (Assignment Triple): An assignment triple $a_{ijx} =< w_i, t_j, s_x >$ indicates that SC platform assigns worker w_i to complete task t_j at the timestamp s_x.

Once a worker w_i is assigned to a specific task t_j by platform, he/she immediately move to the destination, location l_j. We note the travel time of this assignment triple as follows:

$$\tau_{travel} = \frac{\mathrm{d}\left(l_i, l_j\right)}{v_i} \tag{1}$$

which indicates the travel time of the assignment triple a_{ijx} starts from the worker w_i receives the task t_j, and ends when he/she arrives the location of the request. And the $\mathrm{d}\left(l_i, l_j\right)$ means a distance between location l_i and l_j in a 2D Euclidean space or a road network.

B. PROBLEM DEFINITIONS

Definition 4 (Batch Size): Batch size is denoted by bs, indicating that the platform performs task assignment every bs time units, and bs_i means the time interval between the $(i-1)-th$ and the $i-th$ task assignment.

Definition 5 (Waiting Time): A total waiting time τ_{ijx} indicates the time interval from request t_j appears on the platform to the assigned worker w_i reached the working position.Obviously we can notice that:

$$\tau_{ijx} = \tau_{batch} + \tau_{travel} \tag{2}$$

which means the total waiting time of an assignment consists of τ_{batch}, the time period it waits for platform's allocation when current batch is split and τ_{batch}, the travel time worker spends on road.

So the average waiting time of a given time interval T can be calculated as follows:

$$\mathrm{avg}_{a_{ijx\in K}} = \frac{\sum a_{ijx\in K}\,\tau_{ijx}}{|K|}, \tag{3}$$

where |K| is the number of assignment triple set K the platform made during this period.

Definition 6 (Dynamic Batching Model): In a given time period T, an unknown number of workers and tasks will appear dynamically on platform in random order. The platform needs to split the period T into n batches, whose size $(bs_1, bs_1, \ldots, bs_n)$ can be different. At the beginning of every batch, the platform makes a task assignment instance set K, with all available workers W and tasks R at that time. Once this batch ends, workers and tasks involved in K will be removed from W and R, respectively. In the following bs_i time units, M and N may be updated due to the appearance of new workers and new tasks, or the logout of workers and the cancellation of tasks. By setting a series of batch sizes with a certain strategy, the platform can achieve a minimum average waiting time.

IV. MAB-BASED SOLUTION

In this section, we first introduce multi-armed bandit algorithm briefly, then use one of MAB strategies, ε-greedy to select the suitable batch size.

A. MULTI-ARMED BANDIT

Multi-armed bandit (MAB) for a gambler, a typical problem in reinforce learning, originally described by Robins [10], is to decide which arm of a K-slot machine to pull to maximize his total reward in a series of trials.

A K-armed bandit, is like a multi-lever traditional slot machine while the reward of each lever is unknown initially. Our target is select a strategy series to acquire the most revenue. Since each selection from k arms is an independent event, another constraint of the K-armed Bandit problem is to maximize the one-step reward, that is, to ignore future rewards. There are two simple strategies:

- exploration-only: Allocate each lever with an equal probability to try, in which way the reward of each arm can be estimated well, but many opportunities to choose the optimal arm will be lost.
- exploitation-only: Only choose the lever with the currently highest reward, in which way the unexplored best arm may be ignored continuously.

To alleviate the obvious drawbacks of two above methods, Watkins [11] firstly proposes ε-greedy strategy, probably the simplest and the most widely used algorithm to solve the bandit problem [12]. In this algorithm, the MAB explores with ε-frequency, and exploit with frequency $1-\varepsilon$. By adjusting the exploration rate ε, users can achieve the trade-off between these two basic strategies and obtain an ideal outcome. In this paper, we will use this efficient algorithm to make next batch size selection every time platform finishes a batch.

B. MAB-BASED SELF-ADAPTIVE BATCHING ALGORITHM

In our batch split mechanism, using the property of Markov of the problem [8], we first model the batch splitting decision as process of selecting appropriate batch size decision from candidate set, then present a reasonable solution based on MAB algorithm.

1) BASIC IDEA

As aforementioned, we accomplish task assignment on batch mode. After each batch is split, we transfer the initiate problem into a bipartite graph matching problem and solve it with existing efficient method (e.g., KM algorithm). The main problem we must solve is to split batches at the right time.

Since the batch splitting process can be modeled as a Markov decision process(MDP) [8], where the current state and actions will directly influence the state of the next phase, we can use multi-armed bandit to complete these sequential decision due to its superiority in decision-making aspect.

Considering the optimal object is average waiting time of spatial users, we give a maximum waiting time value acceptable to most users. Then average this time period into several batch sizes as alternatives, and model them as multi-arms of MAB. By observing current platform

Algorithm 1 MAB-Based Self-Adaptive Batching Algorithm

```
Input: Batch size array $A$, the timestamp set $S$
Output: An allocation result $M$
initialize exploration rate $\varepsilon$ as $\alpha$;
Reward array $R \leftarrow$ [0 for $bs$ in $A$];
while incoming timestamps do
    if $random(0, 1) \leq \varepsilon$ then
        $bs = RandomChoice(A)$;
    else
        $bs = A[min(R)]$;
    end
    set the current timestamp as $s_x$;
    for workers $W$ and tasks $T$ in $B_{s_x - s_{x+bs}}$ do
        allocate $w_i$ to $t_j$ with task assignment algorithms;
        $M \leftarrow M \cup < w_i, t_j, s_x >$;
        calculate $avg_{a_{ijx \in B_{s_x - s_{x+bs}}}}$;
    end
    update corresponding item in $R$;
    current timestamp $\leftarrow s_{x+bs}$;
    if $|\sigma_{R_{s_x}} - \sigma_{R_{s_{x+bs}}}| > \beta$ then
        if $count > \theta$ then
            $\varepsilon \leftarrow \varepsilon - \lambda$;
            reset $count$;
        else
            $count++$;
        end
    end
end
return $M$
```

performance, dynamically adjust MAB's next action (selecting a best batch size from candidates). Through these processes, we continuously split batch to reach the average waiting time optimization.

2) MODELING

The task we aim to do with ε-greedy MAB algorithm is to repeatedly select a batch size $bs_i \in \{bs_1, \ldots, bs_k\}$ when each batch begins, in the way of exploring or exploiting.

We model our candidate batch size set $\{bs_1, \ldots, bs_k\}$ with multi arms of MAB, so the splitting batching procedure can be regarded as selecting a lever with unknown revenue.Similar to the MAB algorithm, the average waiting time a_i can be used as score of each candidate batch size bs_i. In our model, once choosing a candidate batch size bs_i, we compute the average waiting time of all assignment pairs in that batch and update a_i as follow:

$$a_i = \frac{a_{i1} + a_{i2} + \cdots + a_{ij}}{j} \tag{4}$$

where a_{ij} is the average waiting time (awt) of assignment pairs in candidate bs_i, when it was chosen for batching for the j-th time. And j indicates the time this batch size option has totally been chosen, while a_i is the current awt of bs_i.

In ε-greedy MAB algorithm, we decide to explore or exploit according to the exploration rate ε, which can also be modelled in our mechanism. If the current strategy is exploitation, we pick the candidate batch size that has the least awt value. If explore, we need to randomly select an alternative batch size. The choice of strategy always bases on ε and whatever action (explore/exploit) we choose, the chosen option's awt value has to been updated after the assignment of this batch is finished.

In order to better adapt to dynamic scenarios, we not only update the reward of each candidate batch size according to real-time platform performance, but also adjust the exploration rate ε dynamically, which is an innovation of classical MAB(ε-greedy) algorithm. If the exploration rate ε is too low, the multi-armed bandit will always tend to choose those high-score candidates. If the exploration rate ε is too high, the performance probably get unsteady.

Therefore, we design two reasonable criteria in exploration rate ε's adjustment. When a certain batch size is chosen whose score is far better than others, and after its batching assignment, updated awt value still lower than the minimum of all previous data, the continuous exploitation of it is wiser than aimless exploration. So the first parameter is the mean variance (var) of all current awt value list of total given batch sizes. The second parameter, the minimum awt value of current candidate list can indicates the superiority of current choice if it is smaller than the counterpart in last batching decision.

When deciding whether to explore or exploit, we will use a random function to generate a value between 0 and 1 to compare with the value of ε. Considering the problem of dynamically adjusting the value of ε, we propose a MBA-based reward array R which corresponds to the batch size array A to record the target value we set for each batch size, which is the average waiting time of users. In our framework, each time the $avg_{a_{ijx \in B}}$ is calculated, we will figure out the average of historical average waiting time and update the reward array R with the latest value. In order to successfully monitor the effectiveness of the current batch size, we introduce two variables, the count value and the variance of R, while the σ represents the variance of the reward array R. For the specific process, please refer to lines 16-23 in Algorithm 1.

V. EXPERIMENTAL EVALUATION

In this section, we present the experimental results on synthetic and real-world datasets with the proposed MAB-based algorithm.

A. EXPERIMENTS SETUP

We use both synthetic and real-world datasets in our experiments.

The dataset we used in our experiments is from Didi Chuxing. Specifically, this real-world dataset contains detailed time and location information of drivers' ridesharing orders in Second Ring Road of Xi'an, which are retrieved from a full day time period with a latitude of 34.20531° to 34.28022°

and a longitude of 108.9111° to 108.9986°. Each tuple in this dataset is a representative of corresponding order's spatio-temporal data, which consists of current timestamp, the latitude/longitude of the passenger or driver's position. In order to facilitate our model, we assume the current timestamp and the latitude/longitude of the passenger or drivers to be the time and location of a task or a worker respectively. Considering the situation that there are more than 100,000 orders in one day's dataset, we only select representative time periods during the day. In this paper, the time periods we select are 9:00-11:00, 14:00-16:00 and 20:00-22:00. With the purpose of further proving the practicability of the adaptive batching mechanism proposed in this paper, we adjusted the supply/demand ratio by randomly increasing the number of workers or users.

For synthetic data sets, we generate workers/tasks that constantly join the spatial crowdsourcing system. We generate a group of worker and task sets every ten seconds, creating a total of 720 groups of data which run through two hours. The position of the workers is randomly distributed in a region of a latitude of 34.20531° to 34.28022° and a longitude of 108.9111° to 108.9986°, with a normal distribution of speeds between [4.52, 25.85] m/s. In addition, we also divided the supply and demand ratio into three group: 0.5, 1, and 2, and set the $(6k, 12k)$, $(12k, 12k)$, $(12k, 6k)$ worker-to-task ratio for each group.

We conduct our mechanism presented in this paper with the following task assignment methods.

- *Greedy algorithm (GR).* This is a simple method in batch mode. The main operation it performs is to circularly match the new arriving task to the nearest worker who is idle in different batches.
- *KM algorithm (KM).* It is an excellent algorithm which developed mainly for binary graph matching problem, therefore we modified it to apply to our problem model.

Since the waiting time of tasks is closely related to user experience and user experience is the primary consideration in our experiment, the average waiting time should be taken into consideration when evaluating the algorithms mentioned above. Also, all the algorithms are evaluated in terms of running time (measured by seconds) and memory cost (measured by MiB). In this article, we have selected some of the result graphs as a representative. Finally, the experiments in this paper were performed on an Intel(R) Core(TM) i7-8700 3.2GHz CPU and 8GB main memory, and the algorithms were implemented in Python.

B. EXPERIMENTAL RESULTS

1) IMPACT OF BATCH SIZE

Fig.2 shows the users' average waiting time of different fixed batch sizes. From the overall trend of change, with the batch size increasing, the average waiting time first shows a downward trend, and after a certain degree, there is an upward trend. This is because the initial batch size is too small to accumulate a large number of objects for assignment, as a result, the task may be assigned to workers who are

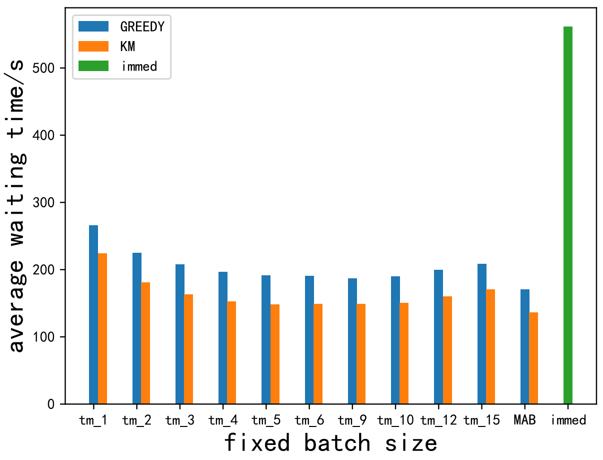
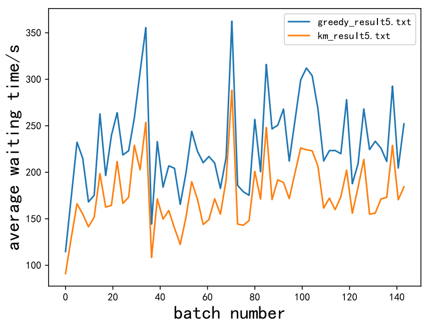


FIGURE 2. **Results on fixed batch size, MAB and immediate allocation.**

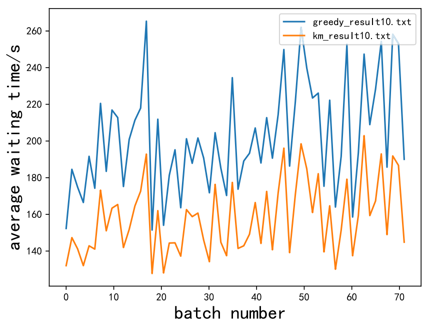
far away thus causing an unreasonable allocation. On the other hand, if the final batch size is too large, even though each user will be more likely to be assigned to a nearby worker, the user will spend too much time waiting for an allocation and users' satisfaction with the platform may drop dramatically. Therefore, there must exist an optimal critical point between these batch sizes. In the example illustrated above, 60s is the best performing fixed batch size. However, there is no fixed optimal batch size under rapidly changing situations. The reason is that in different periods of the day, there can be different traffic conditions, order supply, demand ratio and so on. We further discovered that under the same batch size, the average waiting time of the KM algorithm is less than that obtained by the Greedy algorithm. Real-time allocation of algorithms is particularly poor. What's more, the MAB approach we proposed can get an excellent result. According to the results presented in Fig.2, the average waiting time obtained by the MAB algorithm is significantly better than the best effect achieved by the fixed batch size, which reduces the average waiting time by $10\% - 15\%$.

2) IMPACT OF BATCH NUMBER

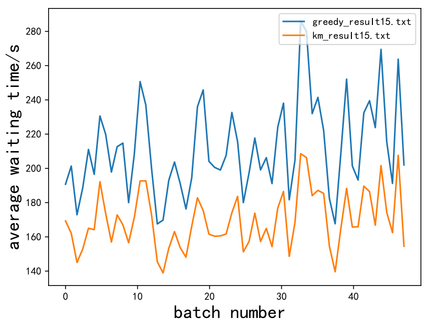
We further observe the changes in the average waiting time over each time period. In our experiment, we randomly select the fixed batch size whose length is 50 seconds,100 seconds, 150 seconds and MAB to make the comparison. From the result, we can find that at the beginning, both fixed batch size and MAB have high fluctuations, and even MAB presents a poor performance. However, as the time goes, the fluctuation of the average waiting time of the user under the fixed batch size does not showed signs of abating as in the MAB, resulting in a higher total average waiting time. The reason is that the exploration rate in MAB will gradually become smaller and tend to be stable over time without being fixed. When the supply-demand ratio changes, MAB can still change the exploration rate reasonably through feedback information to minimize the users' average waiting time. Also, the fixed batch size method does not improve performance of GR or KM. It can be seen that from Fig.3(a)(b)(c) that the curves of the two algorithms are parallelly driven.



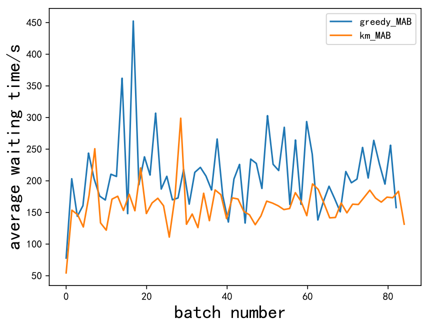
(a) fixed batch size of 5



(b) fixed batch size of 10

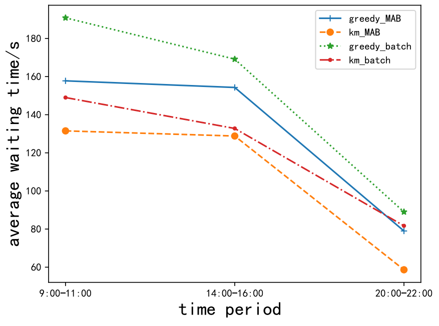


(c) fixed batch size of 15

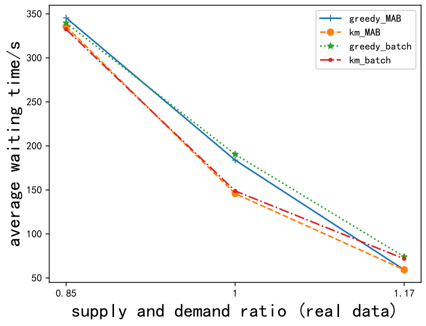


(d) MAB-based approach

FIGURE 3. Results on the increase of batch number.



(a) varying time period



(b) varying supply/demand ratio

FIGURE 4. Results of varying time period and supply/demand ratio on real-world data.

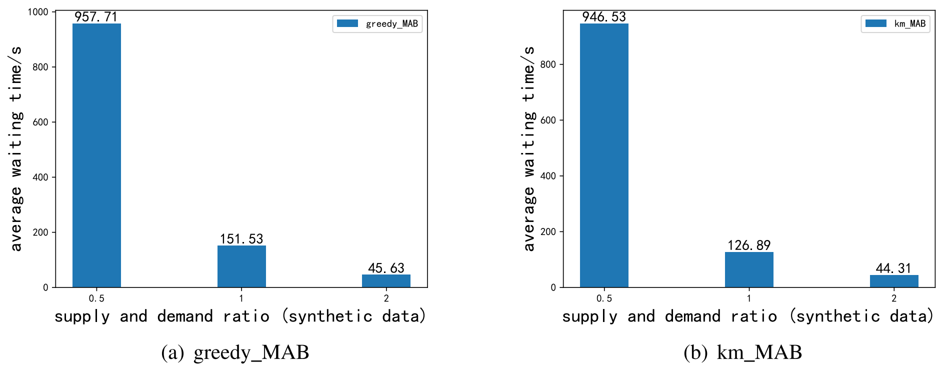
While in Fig.3(d), as the GR algorithm advances over time, the average user waiting time will gradually decrease, even with the KM algorithm, thus reflecting the ability of our mechanism to optimize the performance of existing algorithms.

3) IMPACT OF PERIOD OF TIME

Since traffic conditions and traffic flow throughout the day change over time, we extracted data from three time periods of the day for analysis. Three time periods are 9:00-11:00, 14:00-16:00 and 20:00-22:00 respectively. There are 12379 workers and 27539 tasks in 9:00-11:00, 15641 workers and 50305 tasks in 14:00-16:00, and 12306 workers and 11432 tasks in 20:00-22:00. The demand/supply ratio are 2.225, 3.216, 0.929 corresponding to three different period of time. The results of four different algorithms are represented in Fig.4(a). We can conclude from the figure that when the supply exceeds demand, the average waiting time of the user is far less than the situation of short supply. Therefore, we further observe the impact of different supply and demand ratios on the average waiting time of users on real data sets and synthetic data sets.

4) IMPACT OF SUPPLY AND DEMAND RATIO

Different supply and demand ratios also have an important impact on the average waiting time of users. When supply exceeds demand, the workers assigned to users are likely to be better, and the experience value of users is better because there are more candidate workers to be assigned to users. On the contrary, if the supply is less than the demand, because of the shortage of workers, the system does not have much choice when assigning workers to users, and the



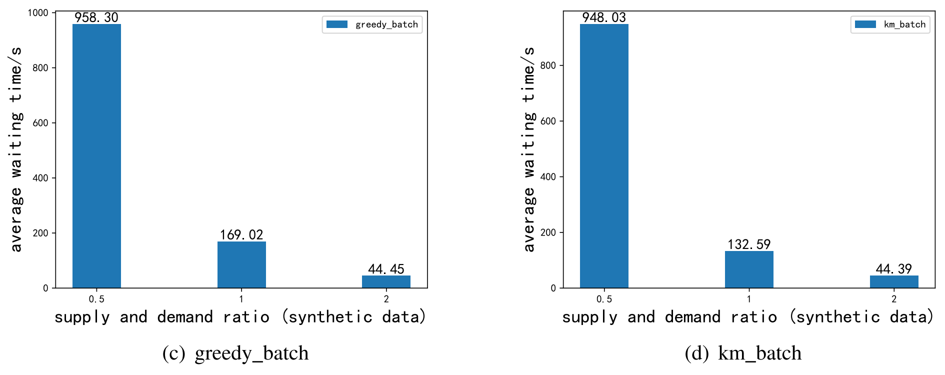


FIGURE 5. Results of varying supply/demand ratio on synthetic data.

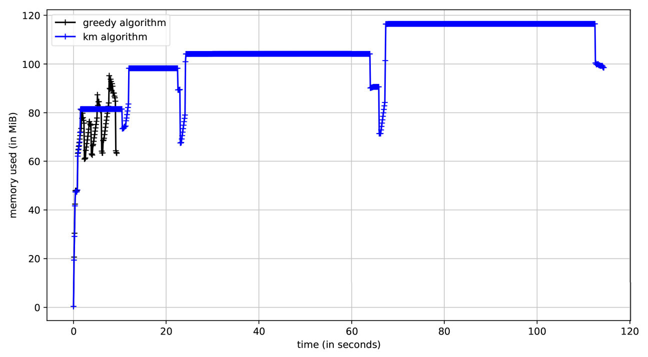


FIGURE 6. Time and memory cost.

final distribution result may be unsatisfactory. Fig.4(b) shows the result in real-world data experiment. When the supply is less than demand, the order of the average waiting time from small to large is km_MAB, km_batch, greedy_MAB, greedy_batch. However, the difference between them is not large. When the supply and demand are equal, the MAB method is significantly better than the batch_based method. When the supply exceeds demand, the time of greedy_MAB is better than km_batch, only a little more than the time of km_MAB. Looking at these three scenarios, the average waiting time of km_MAB is always the best, which further confirms the validity of our proposed MAB.

We set the supply-demand ratio on the artificial data to 0.5, 1, 2, which represent the three situations of supply less than demand, supply and demand balance, and over-demand. Fig.5 show the results of four algorithms at three supply and demand ratios. We can find that the results based on the MAB algorithm are superior at almost all supply and demand ratios. However, when the supply and demand ratio is too large or too small, such as 0.5 or 2 in this example, the improvement of the result based on the MAB algorithm is not obvious. For example, if the supply-demand ratio is 2, greedy_MAB is worse than greedy_batch.

5) TIME AND MEMORY COST

Since immediate allocation algorithm consumes very little memory and time, we will neglect its running cost and just focus on the Greedy algorithm and KM algorithm. As the Fig.6 shows, when comparing with KM algorithm, greedy algorithm not only runs faster, but also consumes less memory. Under the same dataset of workers and tasks, KM algorithm consumes about 1.2 times as much memory as Greedy algorithm, and runs nearly 10 times longer than Greedy algorithm. However, the average waiting time of users through the KM algorithm will be less than that of Greedy, as we describe above.

VI. CONCLUSION

In this paper, we propose a self-adaptive batching mechanism for spatial crowdsourcing, which better take care of the user experience and take into account a more realistic situation. Considering the nature of the mechanisms we design, some traditional task assignment methods can also be combined with our self-adaptive batching mechanism, which can improve the efficiency of dynamic task allocation in online mode as a whole, and also shows good compatibility. Based on this framework, we designed a dynamic batch size adjustment scheme based on multi-arm bandit algorithm, a reinforcement learning based algorithm that infer the next most appropriate batch size based on historical data. We also propose users' average waiting time to be the optimizing objective, thus improving the user experience and fitting the needs of real-world situation. Extensive experimental results of actual and synthetic data sets demonstrate the effectiveness and effectiveness of our mechanisms.

REFERENCES

[1] Z. Huang, N. Kang, Z. G. Tang, X. Wu, Y. Zhang, and X. Zhu, "How to match when all vertices arrive Online," in *Proc. 50th Annu. ACM SIGACT Symp. Theory Comput.*, Jun. 2018, pp. 17–29.

[2] U. U. Hassan and E. Curry, "A multi-armed bandit approach to Online spatial task assignment," in *Proc. IEEE 14th Int. Conf. Scalable Comput. Commun. Assoc. Workshops*, Dec. 2014, pp. 212–219.

[3] L. I. Besaleva and A. C. Weaver, "CrowdHelp: Application for improved emergency response through crowdsourced information," in *Proc. ACM Conf. Pervasive Ubiquitous Comput. Adjunct Publication*, Sep. 2013, pp. 1437–1446.

[4] D. C. Brabham, "Crowdsourcing as a model for problem solving: An introduction and cases," *Converg., Int. J. Res. New Media Technol.*, vol. 14, no. 1, pp. 75–90, Feb. 2008.

[5] H. Dang, T. Nguyen, and H. To, "Maximum complex task assignment: Towards tasks correlation in spatial crowdsourcing," in *Proc. Int. Conf. Inf. Integr. Web-Based Appl. Services*, Dec. 2013, p. 77.

[6] X. Chen, X. Wu, X.-Y. Li, X. Ji, Y. He, and Y. Liu, "Privacy-aware high-quality map generation with participatory sensing," *IEEE Trans. Mobile Comput.*, vol. 15, no. 3, pp. 719–732, Mar. 2015.

[7] Y. Chon, N. D. Lane, Y. Kim, F. Zhao, and H. Cha, "Understanding the coverage and scalability of place-centric crowdsensing," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, Sep. 2013, pp. 3–12.

[8] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *Proc. 35th IEEE Int. Conf. Data Eng. (ICDE)*, Macao, China, Apr. 2019, pp. 1478–1489.

[9] S. Raghavendra, B. Gummidi, X. Xie, and T. B. Pedersen, "A survey of spatial crowdsourcing," *ACM Trans. Database Syst.*, vol. 44, no. 2, Apr. 2019, Art. no. 8.

[10] H. Robbins, "Some aspects of the sequential design of experiments," *Bull. Amer. Math. Soc.*, vol. 58, no. 5, pp. 527–535, 1952.

[11] C. J. C. H. Watkins, "Learning from delayed rewards," Tech. Rep., 1989.

[12] J. Vermorel and M. Mohri, "Multi-armed bandit algorithms and empirical evaluation," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 2005, pp. 437–448.

[13] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *ACM Trans. Spatial Algorithms Syst.*, vol. 1, no. 1, Aug. 2015, Art. no. 2.

[14] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 997–1008.

[15] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic Online matching in real-time spatial crowdsourcing," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 1009–1020.

[16] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 49–60.

[17] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *Proc. Int. Conf. Manage. Data*, Jun. 2018, pp. 773–788.

[18] Z. Chen, P. Cheng, Y. Zeng, and L. Chen, "Minimizing maximum delay of task assignment in spatial crowdsourcing," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1454–1465.

[19] L. Li, J. Fang, B. Du, and W. Lv, "Spatial bottleneck minimum task assignment with time-delay," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2019, pp. 387–391.

[20] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online task assignment for crowdsensing in predictable mobile social networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 8, pp. 2306–2320, Aug. 2016.

LAI QIAN is currently pursuing the bachelor's degree with Soochow University, Suzhou, China. His research interests include information security, cloud computing, spatial databases, and recommender systems.

GUANFENG LIU received the Ph.D. degree in computer science from Macquarie University, Sydney, Australia, in 2013, where he is currently a Lecturer with the Department of Computing. His research interests include graph database, trust computing, and social computing.

FEI ZHU is currently pursuing the M.S. degree with Soochow University, Suzhou, China. Her research interests include data mining, spatial crowdsourcing, deep learning, and cloud/service computing.

ZHIXU LI received the B.S. and M.S. degrees in computer science from the Renmin University of China, Beijing, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from the University of Queensland, in 2013. He used to work as a Research Fellow at King Abdullah University of Science and Technology. He is currently an Associate Professor with the Department of Computer Science and Technology, Soochow University, Suzhou. His research interests include data cleaning, big data applications, information extraction and retrieval, machine learning, deep learning, knowledge graph, and crowdsourcing.

YU WANG is currently pursuing the M.S. degree with Soochow University, Suzhou, China. His research interests include data mining, big data applications, spatial crowdsourcing, and cloud/service computing. He is a member of CCF.

AN LIU received the Ph.D. degree in computer science from the City University of Hong Kong (CityU) and University of Science and Technology of China (USTC), in 2009. He is currently a Professor with the Department of Computer Science and Technology, Soochow University. His research interests include spatial databases, crowdsourcing, recommender systems, data security and privacy, and cloud/service computing.