A Neural-Based Bandit Approach to Mobile Crowdsourcing

Shouxu Lin
Carnegie Mellon University
Pittsburgh, PA, USA
shouxul@andrew.cmu.edu

Yuhang Yao
Carnegie Mellon University
Pittsburgh, PA, USA
yuhangya@andrew.cmu.edu

Pei Zhang
University of Michigan
Ann Arbor, MI, USA
peizhang@umich.edu

Hae Young Noh
Stanford University
Stanford, CA, USA
noh@stanford.edu

Carlee Joe-Wong
Carnegie Mellon University
Pittsburgh, PA, USA
cjoewong@andrew.cmu.edu

ABSTRACT

Mobile crowdsourcing has long promised to utilize the power of mobile crowds to reduce the time and monetary cost required to perform large-scale location-dependent tasks, e.g., environmental sensing. Assigning the right tasks to the right users, however, is a longstanding challenge: different users will be better suited for different tasks, which in turn will have different contributions to the overall crowdsourcing goal. Even worse, these relationships are generally unknown a priori and may change over time, particularly in mobile settings. The diversity of devices in the Internet of Things and diversity of new application tasks that they may run exacerbate these challenges. Thus, in this paper, we formulate the mobile crowdsourcing problem as a Contextual Combinatorial Volatile Multi-armed Bandit problem. Although prior work has attempted to learn the optimal user-task assignment based on user-specific side information, such formulations assume known structure in the relationships between contextual information, user suitability for each task, and the overall crowdsourcing goal. To relax these assumptions, we propose a Neural-MAB algorithm that can learn these relationships. We show that in a simulated mobile crowdsourcing application, our algorithm significantly outperforms existing multi-armed bandit baselines in settings with both known and unknown reward structures.

CCS CONCEPTS

• **Information systems → Crowdsourcing**.

KEYWORDS

mobile crowdsourcing, combinatorial bandit, neural networks

ACM Reference Format:
Shouxu Lin, Yuhang Yao, Pei Zhang, Hae Young Noh, and Carlee Joe-Wong. 2022. A Neural-Based Bandit Approach to Mobile Crowdsourcing. In *The 23rd Annual International Workshop on Mobile Computing Systems and Applications (HotMobile '22), March 9–10, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3508396.3512886

HotMobile '22, March 9–10, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9218-1/22/03.
https://doi.org/10.1145/3508396.3512886

1 INTRODUCTION

The recent proliferation of devices in the Internet of Things has sparked new initiatives in so-called smart cities: large-scale deployments of sensors and actuators across urban areas, which can monitor and respond to their environment [16]. These services, however, rely on the ability of multiple "things" to collect and process data throughout a city. Monitoring vehicle speeds or temperature measurements, for example, requires sensors around a city to collect this data and combine it into a city-wide map. Many such tasks are accomplished with crowdsourcing: recruiting many users or devices to complete subtasks of the overall goal, e.g., sensing pollution at different locations [10], reporting traffic conditions to Google Maps, or asking vehicles with digital billboards to display ads relevant to nearby users. While crowdsourcing has long been studied as a means to accomplish a variety of tasks, e.g., mobile sensing or even image labeling or language translation on platforms like Mechanical Turk [8], IoT environments are significantly more heterogeneous and dynamic, leading to new challenges.

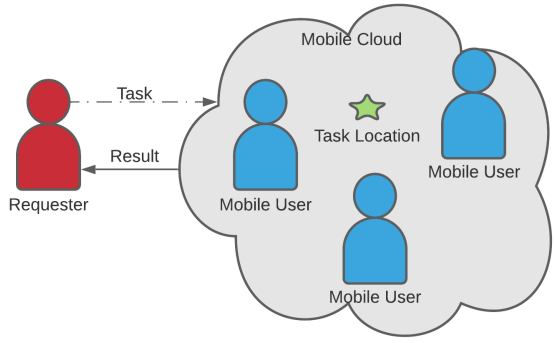

Figure 1: Mobile Crowdsourcing schematic. Requesters send task requests to a mobile cloud, which distributes the tasks to one of several mobile users. The distribution of tasks should consider user availability and suitability for the task.

Task allocation, or deciding which user should accomplish a given task, is a core challenge in mobile crowdsourcing for the IoT. Consider a location-dependent mobile task, such as speed and temperature measurement. As shown in Figure 1, mobile workers

may be able to complete the task if they have mobile devices that are within a certain distance around the target location specified by the requester. However, different devices may be more or less suited to completing the task, e.g., some workers may be more willing to travel to the given location than others. Indeed, even the quality of the completed task can vary greatly. Completing individual tasks may also have different impacts on the overall crowdsourcing task [11]: for example, if the overall task is to construct a temperature map, locations corresponding to heat islands may be more important to monitor than others. These relationships are generally unknown a priori, e.g., worker quality may be hard to predict, which makes the task allocation problem difficult.

One can pose the task assignment problem as a **Multi-Armed Bandit (MAB)** problem, where a decision maker decides the assignment of a mobile task based on its estimation of the quality of mobile workers (called "arms"). The goal is to assign each task to promising workers and thus maximize the cumulative quality of all tasks (called "rewards"). To accommodate many real-life mobile crowdsourcing settings, one may extend the standard MAB to a Contextual Combinatorial Volatile MAB **(CCV-MAB)** problem, where (i) each arm is associated with known context information that can be used to estimate the reward of pulling this arm, (ii) a group of arms (super arm) needs to be selected in each round, rather than a single arm, and (iii) the set of arms available in each round may vary over time. However, existing works impose strong restrictions on the problem setting, which usually cannot be satisfied in practice. We propose Neural-MAB, a solution algorithm that can work without such restrictions. Beyond crowdsourcing, Neural-MAB may be applied to other problems in the CCV-MAB framework, e.g., distributing subtasks in a mobile application across a network of heterogeneous IoT devices.

In this paper, we study a CCV-MAB problem, which combines all of the MAB extensions mentioned above. We design **Neural-MAB**, an algorithm which utilizes one neural network to predict the reward of each single arm, and another to select a super arm. A wide range of mobile crowdsourcing problems can be converted to this general framework and thus be solved by our solution. Specifically, each candidate mobile worker is viewed as an arm in the CCV-MAB problem. Assigning a task to a group of workers (e.g., asking a group of workers to collect pollution measurements at specific locations in a city) can then be converted to the problem of assigning a super arm to each task, with the aim to maximize a pre-defined utility of the assignment (e.g. the quality of the task or a quality-cost ratio). This utility depends on the workers chosen but would generally not be known at the time of task assignment.

The **novelty** of our work lies in four contributions.

- Our formulation is similar to the utility-oriented task assignment problem in crowdsourcing, which matches tasks to workers with the aim to maximize some utility (reward) [21]. Unlike prior works (e.g. [12, 22]), however, we suppose that (i) the utility of assigning a job to a worker is unknown, and (ii) each job is assigned to multiple workers, with its utility depending on each constituent worker.
- We are the first to demonstrate the usefulness of neural networks in combinatorial bandit settings to the best of our knowledge. [24] proposed a similar algorithm for contextual bandits, but does not consider combinatorial arms. Our framework trains the neural network in an online manner, which is particularly challenging in combinatorial settings as the number of super arms, i.e., combinations of individual arms, can be large.
- Neural-MAB imposes no restrictions on the relationships between the reward of each arm and its corresponding context features (called the "single" reward function), or the reward of a super arm and the rewards of its constituent arms (called the "overall" reward function). Prior works assume linear [18] or submodular [4] relationships, which may not hold in practice, e.g., if the context consists of categorical or textual data.
- Neural-MAB does not rely on an (approximation) oracle that can pick the optimal super arm given the rewards of each single arm, as assumed in previous works [4, 17, 18]. Such an oracle requires knowledge of the overall reward function, which may be unknown a priori, e.g., the quality of workers' tasks may depend on the overlap between their data, which is unknown before the data is collected. Experiment results showed that Neural-MAB, with no knowledge about the overall reward function, can outperform existing algorithms with oracle knowledge.

The rest of the paper is organized as follows. Section 2 introduces problem definition and reviews some existing approaches. Section 3 describes the details of the proposed algorithm. Section 4 evaluates the performance of the designed algorithm by applying it to a crowdsourcing application. Section 5 concludes the paper and discusses future works.

2 FORMULATION AND RELATED WORK

In this section, we first introduce the background of several extensions of the standard MAB problem, which the CCV-MAB problem considers. We then define the mobile crowdsourcing problem formally, and show how it can be formulated as a CCV-MAB problem. Next, we introduce our notations. Finally, we briefly walk through the works on MAB, and then highlight the differences between our algorithm and existing ones.

2.1 Background

Contextual MAB is an extension of the standard MAB problem [13], which supposes that the decision maker also observes some context information of each arm to choose in a round. The decision maker then selects an arm by jointly considering the context and the rewards of the arms chosen in the past [3]. In the mobile crowdsourcing setting, the context may be a worker's expertise, which the decision maker can use to predict the quality of each mobile worker. **Combinatorial MAB** is another extension of the standard MAB problem. In many real-world scenarios, a set of arms (called the "super arm") needs to be chosen in each round. For example, we might want to allocate a single task to multiple mobile workers. The challenge comes from the fact that the overall reward of the super arm is not simply a linear function of every single arm, but also depends heavily on the relationship among them [5], e.g., the final quality of a task can be the best task completed by

selected workers. To handle worker mobility, we further consider a **Volatile MAB** framework, in which the set of arms to choose from in each round may vary over time. In the mobile crowdsourcing setting, the candidate mobile workers (arms) for a mobile task at a specific location must be within a certain distance around the target location. Due to the mobility of workers, the candidate workers for tasks at different locations or even tasks at the same location but different times will vary a lot. The CCV-MAB problem we consider contains all the characteristics mentioned above.

2.2 Problem Formulation

In the context of mobile crowdsourcing, we may view candidate mobile workers as arms. We suppose a set of mobile tasks arrives in each round. For each mobile task arriving in each round, the available arms (candidate mobile workers) may change dynamically. For example, a "task" may represent the collection of traffic data in a given location, with "workers" representing vehicles that can collect such data. We define the reward of choosing an arm (i.e., assigning a mobile task to this mobile worker) as the expected quality of the task that will be completed by this mobile worker. For example, the quality of a traffic data collection task might depend on how long the user monitors traffic: monitoring for one minute rather than one second will give a better idea of the average traffic flow. In practice, we have access to some context features about each mobile worker, such as its rating and expertise. Thus, the decision maker can utilize such information to estimate the quality of the task if it is assigned to each mobile worker. The single reward function can then be defined as the expected quality of the task completed by a worker, given the worker's context. For example, it is reasonable to expect a worker with a higher rating and expertise relevant to the mobile task to complete the task with higher quality. Note that the quality of tasks completed by the same worker is not fixed but stochastic, and thus may vary between rounds depending on the realization of the stochastic task quality. Such variation models variation in work quality due to various external circumstances, such as physical and mental conditions.

Each mobile task may be assigned to multiple mobile workers (i.e., selecting a super arm of constituent arms, or workers). Multiple vehicles, for example, may monitor traffic flow in a given location. The overall reward function can be expressed as the relation between the quality of a task completed by a group of workers and the quality of a task completed by each worker in that group. We do not impose any restrictions on the overall reward function. In real-life mobile crowdsourcing problems, there can be various overall reward functions. For example, if the task is to collect traffic data in a specific region, then the overall reward function can be a (i) sum: if each worker only collects data at a specific sub-region with no overlaps between workers, then the total amount of collected data is the sum of the data collected by each worker, or (ii) submodular sum: if multiple workers collaboratively collect data at the same region, then as we add more workers, the overall quality of the collected data increases, but the marginal increase might be decreased as the data collected by multiple workers may overlap.

2.3 Preliminaries

We now introduce mathematical notation formalizing the above formulation. In each round t, $0 \leq t \leq T$, the set of arms (i.e., workers) available for the decision maker to choose from is denoted by A_t. Note that this models the volatile arm setting where the arm set arriving in each round A_t may vary with t. Let $a_t^m \in A_t, 1 \leq m \leq |A_t|$, denote the arms that arrive in round t. The corresponding context features of the arm are represented by the vector $x(a_t^m)$. We let f denote the single reward function, which is an expectation over an unknown distribution parametrized by the associated context feature $x(a_t^m)$. Thus, $f(x(a_t^m))$ represents the expected reward of choosing an arm a_t^m.

In each round, the decision maker will choose a super arm S_t, which is a subset of the arm set A_t. We call each individual arm $a_t^m \in S_t$ a "base arm". We use g to represent the overall reward function, which depends on both the reward of each single arm and the relation between base arms. Thus, the expected overall reward can be expressed as $g(f(S_t))$, where $f(S_t) = \{f(x(a_t^m))|a_t^m \in S_t\}$ is the set consisting of the single reward of each constituent arm. Such a formulation can work with any overall reward function, e.g., either linear, or submodular, or even more complex functions. Usually, a decision maker will have a budget B that limits the maximum number of arms that can be selected, i.e., $|S_t| \leq B$. Note that the decision maker can observe the actual, realized reward of each arm $a_t^m \in S_t$ it has chosen in this round and also the overall reward, respectively denoted by r_t^m (where $\mathbb{E}(r_t^m) = f(x(a_t^m))$) and R_t. These observations can be used to estimate f and g to make better selections in later rounds.

Thus, the objective of the decision maker is to maximize the cumulative overall reward over T rounds. Equivalently, we may view this objective as that of minimizing the cumulative regret, defined as the difference in the cumulative overall reward obtained by the decision maker and that obtained by an optimal decision maker that always selects a super arm S_t^* with the maximum overall reward in each round t. Thus, our goal is to solve

$$\max_{\{S_1,\ldots,S_T\}} \text{reward} = \sum_{t=1}^{T} \mathbb{E}\left[g(f(S_t))\right], \tag{1}$$

or equivalently,

$$\min_{\{S_1,\ldots,S_T\}} \text{regret} = \sum_{t=1}^{T} \left(\mathbb{E}\left[g(f(S_t^*))\right] - \mathbb{E}\left[g(f(S_t))\right]\right) \tag{2}$$

2.4 Related Work

The problem we formulate is similar to the utility-oriented task assignment problem in the **crowdsourcing** literature [21]. However, existing task assignment works generally assume a simpler framework than the one we consider. In particular, they assign each task to only one worker and assume that the corresponding reward is known at the time of task assignment. In this case, the problem can be formulated as a bipartite graph-based problem. Specifically, workers and tasks are represented by the vertices in a bipartite graph, and the reward of assigning each worker to a task can be represented by the weight of the edge between them. The problem is then converted to that of obtaining an optimal matching in the

bipartite graph. We instead use a combinatorial bandit setting, as extending the bipartite matching framework to our combinatorial scenario is difficult. If multiple workers are assigned to each task, the overall reward of a task is not simply a linear function of the reward of assigning each single worker to the task, but also depends heavily on the relationship among them. Thus, we cannot use an edge with a fixed weight to represent the reward of a worker's contributions to a task.

The standard MAB problem has been studied for a long time [1, 19, 20]. The popular Upper Confidence Bound solution algorithm, which is the basis of our proposed solution algorithm (Section 3), associates an upper confidence bound (UCB) estimate of the reward to each arm [1]. The arm in each round is chosen accordingly, and the UCB is then updated according to the reward observations.

Several studies on **contextual MAB** problems have been conducted recently [2, 13, 15]. For example, [15] designs a way to compute the confidence bound efficiently in closed form based on the context features. They assume a linear relationship between the reward of an arm and its corresponding context vector. [24] goes further by using a neural network to learn the relationship between context features and reward, followed by a UCB strategy for exploration. MAB is then extended by considering the combinatorial setting. [18] predicts the reward of each base arm based on the associated context features, and then selects the optimal super arm based on an oracle, which takes the predicted reward of each base arm as input and returns the optimal super arm. The assumption that such an oracle exists and the decision maker has access to it, however, is not realistic in our crowdsourcing setting. [9] works on a combinatorial MAB problem with a submodular reward function, but does not take context features into account.

The most closely related works to ours are [4] and [17], which also consider the CCV-MAB problem. In [4], the context space is uniformly partitioned into hypercubes of identical size, and each hypercube represents an arm group. The arms in the same group are considered to have similar rewards, which are estimated by the average observed reward in this group. They further use a greedy algorithm as an approximated oracle to choose a super arm. The greedy algorithm, however, needs the overall reward function to be known, which takes predicted rewards of a subset of arms as input and outputs the overall reward of this super arm. The decision maker may not know this reward function in crowdsourcing settings. [17] extends [4] by evolving from a static discretization to an adaptive discretization of the context space, which then results in a smaller regret bound. As in [4], however, it still requires an approximation oracle that approximately maximizes the overall reward. By contrast, Neural-MAB can work without such an (approximation) oracle.

3 NEURAL-MAB ALGORITHM

In this section, we propose the Neural-MAB algorithm for the CCV-MAB formulation of mobile crowdsourcing (Algorithm 1). Given a set of arms A_t in each round, the algorithm will choose a super arm S_t which it believes can bring the maximum overall reward at this round. This is accomplished in a two-step process. First, it will estimate the reward of each base arm. Second, it will then pick a super arm based on the estimation of the overall reward function.

Algorithm 1 Neural-MAB

1: **for** $t = 1;\ t <= T;\ t = t + 1$ **do**
2: Initialize $S_t = \emptyset$;
3: **for** $b = 1;\ b <= B;\ b = b + 1$ **do**
4: $a_b = arg\max_{a_t^m \in A_t \setminus S_t} \hat{g}(\hat{f}(S_t \bigcup \{a_t^m\}))$;
5: $S_t = S_t \bigcup \{a_b\}$;
6: **end for**
7: Play S_t and observe the actual single rewards $r_t = \{r_t^m | a_t^m \in S_t\}$ and overall reward R_t;
8: $\hat{f}$ = train($\hat{f}, r_t$);
9: $\hat{g}$ = train($\hat{g}, R_t$)
10: **end for**

Single Reward Estimation: The key point of the contextual MAB problem is to predict the single reward based on the context features. However, the methods proposed in existing works rely on restrictions on the single reward function or the context features. For example, [15] and [18] assume a linear relation between the reward and the context features, and [4] and [17] require the reward of each base arm to be *Hölder* continuous in context features. However, such strong restrictions do not apply in many real-world scenarios. In practice, the context features can be categorical (e.g. educational level) or textual (e.g. description of expertise) data, rather than numerical values. Thus, the key idea of our algorithm is to use a neural network $\hat{f}$ to predict the reward of each base arm based on its context features. Note that this method can (in theory) handle any type of single reward function because any functions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity [7]. We demonstrate its ability to learn nonlinear reward functions in practice in Section 4.

We can use the observed single rewards to train the neural network $\hat{f}$ with back propagation to reduce the difference between estimated single rewards and observed values (line 8). Specifically, suppose we have chosen a set of arms S_t. The inputs of the neural network are then the context features of each arm $\{x(a_t^m) | a_t^m \in S_t\}$, the outputs are the estimated single rewards $\hat{f}(S_t) = \{\hat{f}(x(a_t^m)) | a_t^m \in S_t\}$, and the desired outputs are the observed single rewards $r_t = \{r_t^m | a_t^m \in S_t\}$.

Overall Reward Estimation: Solving combinatorial MAB problems is difficult due to the complexity of combinatorial optimization, which is an NP-hard problem in general [23]. Existing works [17, 18] thus assume the decision maker has access to a polynomial-time oracle that chooses an approximately optimal super arm. [4] goes further by designing a greedy algorithm to serve as an approximation oracle. But the greedy algorithm requires the reward function to be known. Neural-MAB removes such unrealistic assumptions by using another neural network $\hat{g}$ to learn the overall reward function, with which we can estimate the overall reward given any super arm and then pick the estimated optimal one.

After we have trained a neural network to approximate the overall reward function, how to select the optimal super arm is still a challenge. A straightforward way is to feed the neural network with all possible subsets S_t of the arm set A_t and choose one which can generate the maximum output value of the neural network.

However, this method has a high computation complexity. Thus, we adopt a greedy approach inspired by [4], suitably modified to work with any unknown overall reward function. Specifically, we select B arms step by step, and at each step, we pick the arm that can bring the maximum increase in (our estimate of) the overall reward (from lines 3 to 6).

4 EXPERIMENTS

In this section, we evaluate the performance of the proposed algorithm on a mobile crowdsourcing problem.

- Scenario: the crowdsourcing service provider needs to assign each location-dependent task arriving sequentially in T rounds to a subset of candidate workers who are within a certain distance around the target location.
- User context: c_1: distance between the user and the task location (generated based on Gowalla dataset), c_2: worker's rating (uniformly sample from [0, 1])
- Sub-task quality completed by a single worker $f(c_1) \cdot (c_2)^2$, where f is a Gaussian probability density function with mean 0 and standard deviation 1.
- Task quality completed by a group of workers: the sum of the quality of each sub-task completed by each single worker.
- Budget constraint: 20 workers per round.
- Performance analysis: Our Neural-MAB achieves > 50% better regret compared to baseline algorithms.

The problem we consider here is similar to the one studied in [17], where a decision maker needs to assign each location-dependent task arriving sequentially in T rounds to a subset of candidate workers who are within a certain distance around the target location. Specifically, a task arrives with a target location in each round t. The location $\in [0, 1]^2$ represents normalized longitude and latitude, which are sampled uniformly from $[0, 1]$. The number of candidate mobile workers $|A_t|$ at each round t is sampled from the Poisson distribution with mean = 200. A worker is associated with two context features, a location that lies in $[0, 1]^2$ and its rating sampled uniformly at random from $[0, 1]$. The worker locations are generated using Gowalla dataset [6], which contains user check-in locations from the social networking platform Gowalla. In each round t, the set of candidate workers A_t is generated by randomly selecting $|A_t|$ of these check-ins without replacement. We assign the location information contained in each check-in to the location of each generated mobile worker. In the MAB formulation, each worker is represented by a base arm a_t^m with a two-dimensional context vector $x(a_t^m) = [c_m^1, c_m^2]^T$, where c_m^1 denotes the normalized Euclidean distance between the worker and the task locations, and c_m^2 represents the worker's rating.

The expected single reward is defined as
$f(x(a_t^m)) = gauss(c_m^1) \cdot c_m^2$,

where *gauss* is the Gaussian probability density function with mean 0 and standard deviation 1. Note that $f(x(a_t^m)) \in [0, 1]$ represents the expected quality of the task finished by the worker, which is decreasing in the distance between the worker and the task, but increasing in worker's rating. Since the performance of the same worker may vary over time, the actual single reward $r_t^m = N(f(x(a_t^m)), 0.05)$ of selecting worker m in round t is sampled from the normal distribution with mean $f(x(a_t^m)$ and standard deviation 0.05. Intuitively, a task is supposed to have a good quality if it is assigned to a worker with a high rating and located closed to the task location. Since we will assign each job to a group of workers S_t, the overall reward function can thus be expressed as $g(f(S_t)) = \sum_{a_t^m \in S_t} r_t^m = \sum_{a_t^m \in S_t} N(f(x(a_t^m)), 0.05)$. Intuitively, the reward of assigning a task to a group of workers will be the sum of the quality of each sub-task completed by each single worker. We need to select $B = 20$ workers to complete the task in each round with the aim to minimize the cumulative regret, which is calculated by the gap between the optimal cumulative reward and the actual cumulative reward.

We believe this simulation can provide insights about the performance of Neural-MAB in real-world crowdsourcing scenarios. The two context features that we consider, which would likely be relevant in practice, are user location and reliability. Our location data are based on the real user location data in the Gowalla dataset, and thus have a realistic distribution. In practice, the current location of a user will significantly affect its willingness to perform a task at a given location. Although the reliability data is synthetically generated, it can be considered as the normalized values of some real-world data. In practice, it is reasonable to expect data to be normalized in order to expedite training and reduce training error [14]. In addition, the single reward function that we define takes randomness into account, which can reflect the fact that a worker can perform better/worse than expected from the (location, rating) data.

As noted in Section 2.4, we cannot directly compare the performance of existing crowdsourcing algorithms to ours. We thus compare the performance of our algorithm with two closely related algorithms, CC-MAB [4] and ACC-UCB [17], which solve the CCV-MAB problem with known reward functions. We also compare to a random baseline in which we randomly choose arms.

We first evaluate the performance of Neural-MAB with the assumption that CC-MAB and ACC-UCB rely on. That is, we assume that there exists an oracle, which is denoted by Neural-MAB (oracle) in the figure. In such a case, Neural-MAB only needs to learn the single reward function and then turns to the oracle to get the optimal super arm based on its estimation of each individual arm's reward. As shown in Figures 2 and 3, Neural-MAB (oracle) can learn the single reward function very quickly and achieve almost zero regret. Both ACC-UCB and CC-UCB, as well as a baseline that randomly assigns workers to tasks, achieve higher regret (that is, their assignments are further from the optimal ones), as shown in Figure 2, and correspondingly lower reward (Figure 3).

We finally consider a more realistic setting where Neural-MAB cannot access the oracle, as the oracle should not be known a priori in real-world settings. However, ACC-UCB and CC-MAB cannot work without the assumed knowledge of the overall reward function and/or an oracle that maximizes it. Thus, we compare the performance of our Neural-MAB algorithm (with no knowledge of the overall reward function) with our baselines (which have oracle knowledge). That is, we compare Neural-MAB with baselines in a setting strongly in favor of the baselines. As shown by Figure 2, although Neural-MAB has a higher cumulative regret at the very beginning as it is learning the overall reward function, it finally outperforms the ACC-UCB (oracle) and CC-MAB (oracle) baselines.

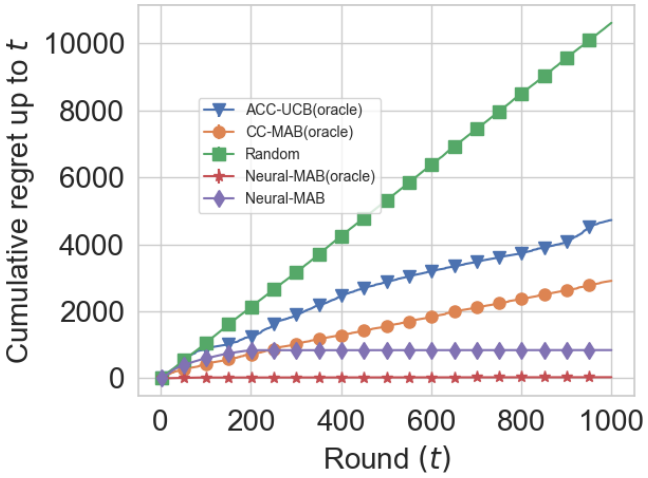

Figure 2: Our algorithm (Neural-MAB) outperforms the baselines of ACC-UCB, CC-UCB, and random assignment of tasks to workers (random). In this figure, we assume that ACC-UCB and CC-UCB have an oracle knowledge of the overall reward function, while Neural-MAB has no knowledge. However, Neural-MAB achieves lower regret over time, showing that it can successfully learn the reward function. We also compare to Neural-MAB with knowledge of the oracle reward function (Neural-MAB (oracle)); it achieves even lower regret than Neural-MAB, due to the additional oracle knowledge.

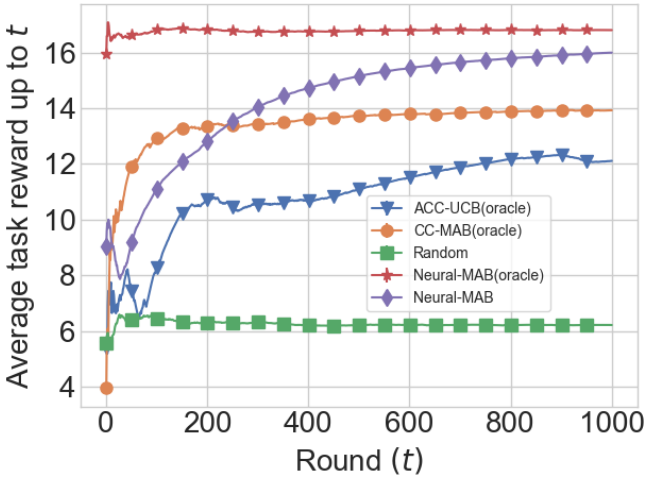

Figure 3: In the same settings as in Figure 2, Neural-MAB achieves higher reward than our baselines, while Neural-MAB with oracle knowledge achieves the highest reward. Towards the beginning of the simulation, CC-MAB (which has oracle knowledge) achieves a higher reward than Neural-MAB, which needs time to learn the reward function, but is overtaken by Neural-MAB around the 200th round.

Neural-MAB(oracle) achieves even better reward due to its prior knowledge of the oracle.

5 DISCUSSION AND CONCLUSION

In this paper, we studied the mobile crowdsourcing problem, which aims to maximize the cumulative quality of sequentially arriving mobile tasks. We first showed that such a problem can be formulated as a CCV-MAB problem. We then designed a Neural-MAB algorithm to solve this problem, which is able to work in a more general environment than existing algorithms, without imposing unrealistic restrictions and assumptions from previous works. We finally evaluated Neural-MAB through a real-world mobile crowdsourcing problem. Experimental results demonstrate that our algorithm performs well compared to existing baselines.

In the future, we plan to compare our results to reinforcement learning methods, which can also handle unknown reward functions. We expect that reinforcement learning may learn more complex assignment policies, but may also take more time to converge.

In addition, we will further apply our solution to more real-world crowdsourcing applications and evaluate its performance on data from these applications. Let us take the restaurant advertising problem as an example. Specifically, we will ask users on a social media platform like Instagram to recommend restaurants. The goal is to increase the sales of the recommended restaurants. To apply our solution, we need to do the following steps. First, we need to convert the problem to a CCV-MAB problem, and clearly define the arm and the task. In this case, each user can be considered as an arm and each restaurant to recommend can be viewed as a task. Second, we need to collect some context information of each arm (e.g. fan base, age, nationality, and occupation of the user) and each task (e.g. reputation, location, type, price of the restaurant). Next, we need to define the single reward and the overall reward. In this scenario, the single reward can be the number of likes/comments of the post that the user writes to recommend the restaurant. The overall reward is simply the increase in the sales of the restaurant. Clearly, the single reward depends on the context information of each user and each restaurant. In particular, if the user is popular and has a large number of fans, then his/her post would tend to attract more likes/comments. In addition, the overall reward is closely related to the single rewards. For example, if a post about recommending a restaurant receives a lot of likes/comments, then the sales of the restaurant will likely increase significantly. Finally, we can apply our Neural-MAB algorithm and evaluate its performance.

In this paper, we consider VM provisioning in an egde system consisting of multiple mobile users communicating with edge servers through a wireless channel. Specifically, mobile users can offload their tasks to a nearby edge server if that edge server already pre-provisions a virtual machine or container with a customized image which contains all dependencies and runtime to execute the task. To explain customized VM provisioning more clearly, we give an example, as shown in Figure **??**. In this figure, two mobile users are in the coverage area of an edge server. Due to the budget constraint and the computation resource limit of the edge server, it only pre-provisions a customized VM for user A. In this case, user A can offload the task to this edge server but the task of user B has to be processed in the remote cloud. Although

Suppose there are K edge servers in the system and each server is represented by m_k with $k \in \{1, 2, .., K\}$. The operational timeline is discretized into time slots {1,2, ..., T}, where T denotes the finite

time horizon. Let U^t denote the users population in each time slot t. The users covered by edge server m_k is denoted by $U_k^t \subseteq U^t$. At the start of each time slot, the users notify the edge servers what customized VMs they require and the edge servers then determine which of them to pre-provision. In this time slot, only the users whose customized VMs have been provisioned can offload their tasks to the edge servers. However, due to the budget constraint and resource scarcity of the edge servers, we can never pre-provision all types of customized VMs in each edge server. Thus, we need to cleverly determine what types of customized VM to pre-provision in each edge server with the aim to minimize the aggregated latency among all the tasks. Let B denote the budget constraint (i.e., the total number VMs we can place in all edge servers), and C_k denote that the capacity constraint of an edge server m_k (i.e. the number of VMs the we can place in each edge server).

In a time slot t, a user $u \in U_t$ can have computation tasks, e.g., gaming, video streaming, and virtual reality tasks, to be offload to the edge serve/cloud server for processing. Since these task are computing-intensive and data-intensive task, and limited computing capacity and battery life, we assume these tasks cannot be handled by mobile devices. Thus, similar as the works in [**? ? ?**], we only consider task processing on either edge servers or cloud servers. Next, we will introduce the single user latency model for Edge/Cloud processing.

5.1 Edge Processing Delay

Note that unlike more powerful devices like desktop or laptop, mobile devices have limited computing capacity and each time a mobile device will execute one application at each time, which is why mobile device manufacturer focus more on the performance of single-processor over multiple processor. We assume that the mobile tasks of a single user in each time slot are of the same type, which can be handled by one customized VM. Thus, in a time slot t, a user $u_n^t \in U_t$ can offload computation tasks j_n^t to a nearby edge server through the one-hop wireless link if a customized VM has been provisioned at that edge server. The task delays are incurred for completing the tasks, which consists of two main parts: transmission delay and computation delay.

Transmission delay: According to the Shannon capacity [**?**], the wireless uplink transmission rate between user u_n^t and edge server m_k at time slot t can be modeled as $r_{n,m}^t = Q \cdot log_2(1 + \frac{W_n G_{n,m}^t}{N+I})$, where Q is the channel bandwidth, W_n is the transmission power of user u_n^t's mobile device, $G_{n,mt}^t$ is the uplink channel gain, N is the noise power, and I is the interference. Since the data size of task result is usually small and therefore it can be neglected, as assumed in works [**? ? ?**]. But this model can be easily extended to incorporate downlink transmission delay. Since mobile devices have limited computing resources and battery life, we assume that the tasks from a single user in each time slot are of the same type. Suppose the input data size of each single task $j_n^t \in j_n^t$ is s_n^t, the transmission delay of offloading all these tasks can then be calculated as $\frac{s_n^t}{r_{n,m}^t}$.

Computation delay: The computation delay depends on the computation workload and the allocated VM's CPU frequency. We assume that the computing resources including CPU and memory at edge sever m_k has been equally separated into C_k VMs. Suppose the CPU frequency of each VM is f_k and a task j_n^t requires σ_n^t number of CPU cycles, the computation delay is $\frac{\sigma_n^t}{f_k}$.

The overall task latency from user u_n^t when processed at edge server m_k is $\delta_{n,k}^t = \lambda_n^t * (\frac{s_n^t}{r_{n,m}^t} + \frac{\sigma_n^t}{f_k})$, where λ_n^t is the number of tasks from user u_n^t.

5.2 Cloud Processing Delay

If a customized VM is not pre-provisioned at edge server m_k for user u_n^t, then it can only be processed by the remote cloud server. Similar to edge processing, the cloud processing delay includes includes wireless transmission delay and cloud computation delay. However, it also incurs the backbone Internet transmission delay. Specifically, the cloud processing latency $\delta_n^t = \frac{s_n^t}{r_0^t} + \frac{s_n^t}{r_1^t} + \frac{\sigma_n^t}{f_0} + \gamma$, where r_0^t, r_1^t, f_0, and γ represents the wireless transmission rate between the user and the cloud server, the backbone Internet transmission rate, CPU frequency of the allocated VM at the cloud server, and the round trip time respectively.

5.3 Delay Reduction

The delay reduction of tasks from a user u_n^t by provisioning customized VM at edge server m_k can be calculated as $\theta_n^t = \delta_{n,k}^t - \delta_n^t$. Suppose $\alpha_{n,k}^t$ is a binary variable which denotes whether or not a customized VM has been pre-provisioned, the delay reduction for this user is then $\alpha_{n,k}^t \cdot \theta_n^t$. Let a set function $G : \{R\} \rightarrow R$ represent the relationship between the delay reduction of each user and the total reduction delay among all the users at the edge server m_k. One may form G as a linear sum function, i.e., $G(\{\theta_n^t | u_n^t \in U_k^t\}) = \sum_{u_n^t \in U_k^t} \theta_n^t$. However, this only holds under the assumption that there is no inference between the VMs at each edge server. As described before, the performance of each VM will decay due to inevitable resource sharing and contention (e.g. shared CPU cache, memory bandwidth, and I/O). Thus, we have $G(\{\theta_n^t | u_n^t \in U_k^t\}) \leq \sum_{u_n^t \in U_k^t} \theta_n^t$, where the equation only holds when there is no inference between tasks, as assumed in works [**? ? ?**].

ACKNOWLEDGMENTS

This research was partially supported by ONR N00014-21-1-2128, NSF CNS-1751075, NSF CNS-2103024, Mobility21 at CMU, AiFi Gift, and UPS Endowment Fund at Stanford.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.
[2] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.
[3] Djallel Bouneffouf and Irina Rish. 2019. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040* (2019).
[4] Lixing Chen, Jie Xu, and Zhuo Lu. 2018. Contextual combinatorial multi-armed bandits with volatile arms and submodular reward. *Advances in Neural Information Processing Systems* 31 (2018), 3247–3256.
[5] Wei Chen, Yajun Wang, and Yang Yuan. 2013. Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning*. PMLR, 151–159.
[6] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th*

ACM SIGKDD international conference on Knowledge discovery and data mining. 1082–1090.

[7] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[8] Xiaofeng Gao, Shenwei Chen, and Guihai Chen. 2020. Mab-based reinforced worker selection framework for budgeted spatial crowdsensing. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[9] Elad Hazan and Satyen Kale. 2012. Online Submodular Minimization. *Journal of Machine Learning Research* 13, 10 (2012).

[10] Linta Islam, Syada Tasmia Alvi, Mohammed Nasir Uddin, and Mafizur Rahman. 2019. Obstacles of mobile crowdsourcing: A survey. In *2019 IEEE Pune Section International Conference (PuneCon)*. IEEE, 1–4.

[11] Shweta Jain, Ganesh Ghalme, Satyanath Bhat, Sujit Gujar, and Y Narahari. 2016. A deterministic mab mechanism for crowdsourcing with logarithmic regret and immediate payments. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. 86–94.

[12] Bala Kalyanasundaram and Kirk R Pruhs. 2000. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science* 233, 1-2 (2000), 319–325.

[13] John Langford and Tong Zhang. 2007. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in neural information processing systems* 20, 1 (2007), 96–1.

[14] Boyi Li, Felix Wu, Ser-Nam Lim, Serge Belongie, and Kilian Q Weinberger. 2021. On feature normalization and data augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12383–12392.

[15] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. 661–670.

[16] Shouxu Lin, Weifa Liang, and Jing Li. 2020. Reliability-aware service function chain provisioning in mobile edge-cloud networks. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 1–9.

[17] Andi Nika, Sepehr Elahi, and Cem Tekin. 2020. Contextual combinatorial volatile multi-armed bandit with adaptive discretization. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1486–1496.

[18] Lijing Qin, Shouyuan Chen, and Xiaoyan Zhu. 2014. Contextual combinatorial bandit and its application on diversified online recommendation. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 461–469.

[19] Herbert Robbins. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58, 5 (1952), 527–535.

[20] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3/4 (1933), 285–294.

[21] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, and Cyrus Shahabi. 2020. Spatial crowdsourcing: a survey. *The VLDB Journal* 29, 1 (2020), 217–250.

[22] Yajun Wang and Sam Chiu-wai Wong. 2015. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *International Colloquium on Automata, Languages, and Programming*. Springer, 1070–1081.

[23] Laurence A Wolsey and George L Nemhauser. 1999. *Integer and combinatorial optimization*. Vol. 55. John Wiley & Sons.

[24] Dongruo Zhou, Lihong Li, and Quanquan Gu. 2020. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*. PMLR, 11492–11502.