Adaptive Holding for Online Bottleneck Matching with Delays

Kaixin Wang* Cheng Long* Yongxin Tong† Jie Zhang* Yi Xu†

Abstract

Bipartite matching between two sets of objects is widely applied in many applications such as crowdsourcing marketplaces, ride-hailing services and logistics. Depending on the applications, different objectives have been proposed, resulting in different matching problems. Among them, one that is recently proposed is *online bottleneck matching with delays* (OBM-D), where the objective is to optimize the maximum cost of matches and the cost of a match depends on when the match is formed (i.e., it is delay-aware). Existing solutions for OBM-D usually adopt a *holding* strategy, which holds the objects involved in a match available for a period so as to reduce the chance that a bad match is formed. Nevertheless, existing holding strategies are all based on human-crafted rules thus cannot adapt to the dynamics of how the objects arrive. In this paper, we propose an adaptive holding strategy which is based on reinforcement learning and develop a method called Adaptive-H on top of the new holding strategy. Besides, we prove theoretical results on how good a randomized algorithm could achieve for the OBM-D problem in terms of competitive ratio. We conduct extensive experiments on both real and synthetic datasets to verify that Adaptive-H outperforms existing algorithms in terms of both effectiveness and efficiency.

1 Introduction

Bipartite matching (also called "assignment problem") is a process of assigning objects in one set to those in another set, and those pairs of objects that are assigned to each other constitute a matching. It is used in various emerging applications, ranging from ride hailing platforms (e.g., assigning requests to drivers by Uber [1], Didi Chuxing [2] and Grab [3]), to food delivery services (e.g., assigning orders to couriers by Ele.me [4] and Grubhub [5]). In these applications, the objects to be matched usually arrive dynamically in an online fashion. Correspondingly, the matching problem is called *online bipartite matching*, where the information about the objects that would appear in the future is not available at the time point when assigning objects.

Depending on the application scenarios, different objectives have been adopted for the online bipartite matching problem [26, 40, 37, 39, 16, 42, 38, 17, 19]. More recently, people studied an objective, which is to minimize the maximum match cost (called *bottleneck cost*), where the cost of a match relies on when it is formed (i.e., it is delay-aware) [15, 30]. For example, when assigning a driver (resp. courier) to a passenger (resp. customer), the cost would become larger if the match is formed later since it means that the passenger (resp. customer) needs to wait longer. The corresponding matching problem is called *online bottleneck matching with delays* (OBM-D). Specifically, given a set of requests and workers, both arriving dynamically, the OBM-D problem is to assign workers to serve requests such that the maximum cost of a match is minimized, where the cost of a match consists of (1) the amount of time the request in a match has waited before the match is formed and (2) the amount of time the worker needs to prepare before it serves the request. In the OBM-D problem, the worst-case match cost is optimized, implying that the match costs tend to be closer to one another and thus forming a fairer matching.

Different from other matching objectives that are based on some *aggregation* over individual matches, the objective of OBM-D is determined by one (or a few) matches, i.e., those matches with the maximum/bottleneck cost. Therefore, extra attention is needed to push the chance of forming a bad match as low as possible. Existing algorithms usually adopt a *holding* strategy, which is to *hold* a match for a certain period once it is formed. The intuition is that the worker involved in the match may be required to cater for another request which would be matched badly otherwise. Existing algorithms implement the holding strategy differently [15, 30]. In [30], the strategy is to hold those matches with costs among the bottom-k where k is a pre-specified parameter. The intuition is that while the costs of these matches increase as time goes by, it is tolerable since they have their costs among the bottom-k and the workers involved in them could potentially be used to cater for future requests that would be matched badly otherwise. This strategy relies on a pre-specified parameter k, which would be fixed throughout the execution of the algorithm. We call the algorithm based on this holding strategy *Fixed-H*. In [15], the strategy is to hold *each* match for a certain amount of time that depends on the match (i.e., it is set to be equal to the amount of time the worker in the match needs to prepare before it serves the request in the match). This strategy does not require any user parameter and the amount of time for holding

*Nanyang Technological University, Singapore, {kaixin.wang, c.long, zhangj}@ntu.edu.sg

†Beihang University, China, {yxtong, xuy}@buaa.edu.cn

varies among the matches. We call the algorithm based on this holding strategy *Variable-H*.

While the holding technique has been verified to work effectively [30, 15], its benefits have not been fully unleashed. Specifically, for both Fixed-H and Variable-H, they are based on human-crafted rules and cannot be adaptive to the dynamics of how requests and workers arrive. For instance, in the taxi service scenario, the dynamics of how passengers make their orders are very different during rush hours and non-rush hours. As shown in our experiments, the performance of the algorithms based on these strategies is far from the optimum. In addition, for Fixed-H, it requires users to set a parameter, which is demanding especially when users do not have experience on how to set it.

In this paper, we propose a new algorithm called *Adaptive-H* for the OBM-D problem, which accumulates requests and workers batch by batch and then performs (offline) bottleneck matching within each batch. In addition, Adaptive-H adopts a holding strategy for matches that have been formed. Different from existing strategies, which are based on human-crafted rules, the holding strategy is learned via reinforcement learning (RL) so as to adapt to the dynamics of how requests and workers arrive. Specifically, we regard the OBM-D problem as a *sequential decision making process*, which is to periodically make a decision on (1) whether it continues to wait for further requests and workers and (2) and if not, which matches to hold (among those that would be formed within the batch via offline bottleneck matching). We then model the sequential decision making process as a *Markov Decision Process* (MDP) and use a standard Q-learning method to learn a policy for the MDP. We carefully design the MDP including states, actions and rewards such that the states capture critical information and are cheap to compute, the action space is manageable and does not incur heavy learning burden, and the rewards are consistent with the goal of the original problem. We defer further details of the RL-based method to Section 3.

In addition, we present theoretical results on the lower bound of the competitive ratio of any randomized online algorithm for the OBM-D problem, thus closing the gap that no such results are known before. Specifically, we prove that no randomized algorithms could provide a competitive ratio better than $\frac{n}{2}$, assuming that we have n requests and m workers ($m \geq n$). In summary, the main contributions of this paper are as follows.

- We propose a reinforcement learning based algorithm Adaptive-H for the OBM-D problem. Compared with existing methods, Adaptive-H uses a new holding strategy, which is data-driven but not based on human-crafted rules.

- We present theoretical results on the lower bound of the competitive ratio of any randomized online algorithm for the OBM-D problem, which closes the gap that no such results for the OBM-D problem are known before.

- We conduct extensive experiments on both real and synthetic datasets to verify that Adaptive-H outperforms existing algorithms in terms of both efficiency and effectiveness. For example, Adaptive-H improves the effectiveness by 32% - 35% and simultaneously runs 70-90 times faster on the real datasets, compared with the state-of-the-art Variable-H.

The remainder of the paper is organized as follows. We first introduce the problem definition and theoretical results in Section 2 and the Adaptive-H method in Section 3. We then present the experiments in Section 4 and the related work in Section 5. Finally, we conclude the paper with future research directions in Section 6.

2 Problem and Preliminaries

2.1 Problem Definition The OBM-D problem was originally proposed in [15]. Here, we review the problem definition and explain the rationale behind for being self-contained. Let R (resp. W) be a set of requests (resp. workers) that arrive dynamically. For a request r_i (resp. a worker w_j), we denote the time step it arrives by $r_i.t$ (resp. $w_j.t$).

Suppose that we assign worker w_j to request r_i at time t. We say that (r_i, w_j, t) is a *match* and r_i and w_j are matched with each other at time t. We consider from the request r_i's point of view the delay before it could be served. First, request r_i has already waited for $(t - r_i.t)$ amount of time without being assigned with any worker. Second, in many application scenarios, once a worker w_j is assigned to request r_i, w_j cannot start to serve r_i immediately, instead, w_j needs to do some preparation before serving r_i. For example, in the taxi service application, once a taxi is assigned to an order from a passenger, the taxi cannot serve the passenger immediately but needs some time to move from its current location to the passenger's pick-up location. We denote by $t(r_i, w_j)$ the amount of time worker w_j needs to prepare before serving r_i. In conclusion, for a match $a = (r_i, w_j, t)$, we define its cost, denoted by $cost(a)$, to be the amount of time r_i waits till w_j starts to serve r_i, i.e.,

$$cost(a) = (t - r_i.t) + t(r_i, w_j) \tag{2.1}$$

The formal definition of OBM-D is given below.

DEFINITION 2.1. *Given a set R of n requests and a set W of m workers, which arrive dynamically ($n \leq m$), the **online bottleck matching with delays** (OBM-D) problem [15] is to form a set of n matches in an online manner such that the maximum cost of a match is minimized, i.e., it finds*

$$M^* \in \left\{ M \mid \arg\min_{M} \max_{a \in M} cost(a) \right\} \tag{2.2}$$

where M is a possible matching with n matches and M^* is one of the optimal solutions.

2.2 Theoretical Results In practice, we can only design *online algorithms* for the OBM-D problem since the input of the problem is exposed piece-by-piece without having the entire input available from the start. The performance of an online algorithm is usually measured by its *competitive ratio*. Specifically, given an online algorithm A for the OBM-D problem, the competitive ratio of A, denoted by $cr(A)$, is defined to be the *greatest* ratio between the objective value of the solution returned by A and that of the optimal solution for all possible instances of the OBM-D problem. That is,

$$cr(A) = \max_{x \in \mathcal{X}} \frac{\mathbb{E}\big[obj(A(x))\big]}{obj(M^*(x))} \tag{2.3}$$

where $\mathcal{X}$ is the set containing all possible instances of the OBM-D problem, $obj(A(x))$ is the objective value of the solution returned by algorithm A on a problem instance x, $\mathbb{E}[\cdot]$ denotes the expectation operator and caters for cases where A is a randomized algorithm, and $obj(M^*(x))$ is the objective value of the optimal solution on the problem instance x. Note that in practice, $M^*(x)$ for a problem instance x is not achievable because it would rely on the full knowledge of the problem input at the very beginning. The lower the competitive ratio is, the better the algorithm is. A common practice is to design an online algorithm with competitive ratio as low as possible.

In [15], it is proved that for the OBM-D problem, no deterministic algorithms can achieve a competitive ratio better than $\frac{n}{\ln 2}$, but no results are known for randomized algorithms. Since our RL-based method, i.e., Adaptive-H, to be introduced in this paper, involves some randomness (for computing states), we are interested in knowing what the competitive ratio boundary that a randomized algorithm might be able to achieve. The result is shown in the following theorem (with the proof provided in a technical report [41]).

THEOREM 2.1. *No randomized algorithm can achieve a competitive ratio better than $\frac{n}{2}$ for the OBM-D problem with n requests.*

3 Reinforcement Learning Approach

In the OBM-D problem, the requests and workers arrive dynamically in an online fashion. We adopt an existing strategy that we periodically (e.g., every 10 seconds) form a batch containing all requests and workers that have arrived but not been matched yet [30, 26, 27]. Different from those existing solutions, which perform an (offline) matching within *every* batch, we aim to achieve an *adaptive* solution that would for each formed batch, take the context of the batch into account and decide whether to perform a matching, and if so, further decide among those matches that have been formed in the batch, which to hold and correspondingly which to return as firmed matches. Specifically, we regard the OBM-D problem as a *sequential decision making process* and model it as a *Markov decision process* (MDP) [34] (Section 3.1), adopt the standard Q-learning method [44] for learning an optimal policy for the MDP (Section 3.2), and then develop an algorithm called *Adaptive-H*, which uses the learned policy for solving the OBM-D problem (Section 3.3).

3.1 The OBM-D Problem Modeled as a MDP We model the OBM-D problem as a MDP, which consists of four components, namely *states*, *actions*, *transitions*, and *rewards* as defined as follows.

3.1.1 States Let t_1, t_2, ... be the time steps, at which we form batches periodically. Let C be the gap between two adjacent time steps, i.e., $C = (t_{k+1} - t_k)$, $k \geq 1$, which captures the frequency that we perform actions and is tunable based on the applications. Let B_k be the batch of requests and workers we form at time t_k ($k = 1, 2, ...$) and $R(B_k)$ (resp. $W(B_k)$) be the set of requests (resp. workers) in B_k.

We denote the state at time t_k as s_k. Intuitively, state s_k should capture the critical information of batch B_k, which is useful for deciding which action to take for B_k at time t_k. As will be introduced later, there are two actions, namely, one to wait till the next time step t_{k+1}, and the other is to perform a matching within batch B_k at time t_k. We identify the following two types of information, which are critical for deciding an action to take at state s_k: (1) *how long the requests in batch B_k have waited for since they arrived by time t_k* and (2) *how long the workers in batch B_k need to prepare for before they can serve the requests*. They are aligned with the two components of the cost of a match, defined in Equation (2.1). Next, we explain how we capture these two types of information one by one in detail.

To capture the first type of information of batch B_k, one straightforward idea is to use the waiting times of all requests in batch B_k starting at the time they arrived and ending at time t_k, but then it would result in a large state space making the model hard to train. Instead, we use the longest waiting time of a request, which we define as the *time span* of batch B_k, denoted by $\theta(B_k)$, i.e.,

$$\theta(B_k) = \max_{r_i \in R(B_k)} (t_k - r_i.t) \tag{3.4}$$

The rationale is that $\theta(B_k)$ has a much smaller space and bounds the waiting times of all requests. A special case is that there exist no requests in batch B_k, and in this case, we define $\theta(B_k)$ to be 0.

To capture the second type of information of batch B_k, a first attempt could be to use the amounts of time that the workers need to prepare for the requests in batch B_k, but then again, this would result in a very large state space making it inefficient to train the model. A second attempt could be to (1) build a weighted bipartite graph between the workers and requests in batch B_k, where for each pair of a worker and a request, there exists an edge with the weight set to be

the amount of time that the worker needs to prepare before it serves the request, and (2) use the maximum weight of a match in the bottleneck matching on the weighted bipartite graph, which we denote by $w(B_k)$, to capture the second type of information. While this idea looks intuitive and would result in a simpler state space, it would result in a high computational workload since it involves a bottleneck matching procedure, which takes at least cubic time [32].

Our solution is to compute a lower bound and an upper bound of $w(B_k)$, each with a cheap procedure, and then use the average of the bounds to act as an estimate of $w(B_k)$, which we denote by $\widehat{w(B_k)}$. Specifically, to obtain a lower bound of $w(B_k)$, for each request r_i in batch B_k, we collect the smallest amount of time that a worker in batch B_k needs to prepare before serving r_i, which we denote by $t_{min}(r_i)$. Note that this procedure is much more efficient than a bottleneck matching procedure. In addition, it could be verified that $\max_{r_i \in R(B_k)} t_{min}(r_i)$ corresponds to a lower bound of $w(B_k)$. To obtain an upper bound of $w(B_k)$, we form a maximum matching within batch B_k randomly and then use the maximum weight of a match in the formed matching as the upper bound of $w(B_k)$, and this could be easily verified since $w(B_k)$ corresponds to the smallest possible weight of a match in a maximum matching within batch B_k by definition. Furthermore, we discretize $\widehat{w(B_k)}$ by dividing it by a bin size, denoted by W, and then feeding the result to a ceiling function. In conclusion, we use $\sigma(B_k)$ (defined as follows) to capture the second type of information of batch B_k.

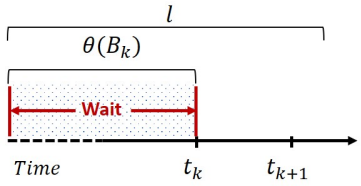
$$\sigma(B_k) = \lceil \frac{\widehat{w(B_k)}}{W} \rceil \tag{3.5}$$

Here, the rationale of using a bin size is to compress the state space.

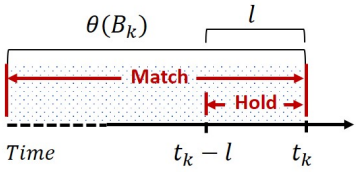
In summary, we define state s_k to be a pair of two integers, namely $\theta(B_k)$, which captures the waiting times of requests before workers are assigned to them, and $\sigma(B_k)$, which captures the waiting times that are due to the necessary preparations by the workers, i.e.,

$$s_k = (\theta(B_k), \sigma(B_k)) \tag{3.6}$$

3.1.2 Actions We denote the action we take at time t_k by a_k. Recall that at t_k, the state is s_k and we need to decide whether to perform a matching within batch B_k or to wait till the next time step. If yes, then we further decide among those matches that have been formed, which to hold and correspondingly which to be returned. A straightforward method to define an action to capture this decision process would need to use at least two values, one to indicate whether to perform a matching or not, and the other for indicating which matches to hold (if it has formed some matches). We achieve the same goal with one value. Specifically, we define an action to be an integer in a range $[0, D]$, where D is a hyper-parameter that could be tuned, and an action $l \in [0, D]$ means to achieve a batch with its time span at least l before we perform a matching. Depending on the time span of batch B_k, i.e., $\theta(B_k)$, we have two cases for an action $l \in [0, D]$.



(a) Case 1: $l > \theta(B_k)$. Wait



(b) Case 2: $l \leq \theta(B_k)$. Match and Hold

Figure 1: Illustrations on the actions

- **Case 1 (Wait):** $\theta(B_k) < l$**.** In this case, batch B_k has the time span strictly smaller than the target one l, and thus the action would be to wait till the next time step t_{k+1}. For illustration, consider Figure 1(a). Note that $\theta(B_{k+1})$ would be at least $\theta(B_k)$.

- **Case 2 (Match and Hold):** $\theta(B_k) \geq l$**.** In this case, the time span of B_k has reached or surpassed the target one l, and thus the action is to perform a bottleneck matching within batch B_k. Furthermore, the action would also be to hold among all formed matches, those with the requests arriving later than $(t_k - l)$ since the batch involving these requests would have the time span less than l, and return all other matches. For illustration, consider Figure 1(b).

In conclusion, the action $a_k = l$ is used as both a trigger for performing a matching (when the time span of batch B_k becomes at least l) and a threshold for controlling which matches to hold (those with arriving times later than some threshold based on l, i.e., $t_k - l$).

3.1.3 Transitions In our process of online matching, given a current state s_k at time t_k and an action a_k to take, the probability that we would observe a specific state s_{k+1} is unknown since it would depend on how the future requests and workers arrive. We note that the method that we use for solving the MDP in this paper, i.e., Q-learning [44], is a *model-free* one, which could solve the MDP even with its transition information unknown.

3.1.4 Rewards We denote by R_k the reward associated with the transition from state s_k to state s_{k+1} after taking action a_k and define it as follows. Suppose $a_k = l \in [0, D]$. In the case that $\theta(B_k) < l$, the action is to wait till the next time step, and thus no matches are made and the maximum cost of the matching formed so far does not change. But since this would increase the costs of those matches, which are formed later on but could possibly be formed at time t_k otherwise, we define the reward to be -1 as a signal of penalty. In the other case that $\theta(B_k) \geq l$, the action is to perform a matching and then hold some matches while returning others. We denote by c_k (resp. c_{k+1}) the maximum cost of a match in the matching before (resp. after) the action a_k is taken. Note that since there is no matching before a_1 is taken, we define c_1 to be the value of the optimal maximum cost under the offline scenario. Also note that the maximum cost of a match in a matching is non-decreasing, i.e., $c_{k+1} \geq c_k$. We further consider two sub-cases depending on c_{k+1} and c_k. In the sub-case that $c_{k+1} = c_k$, we define the reward R_k to be cnt, where cnt is the number of *consecutive* time steps, at which we have $\theta(B_k) < l$ (i.e., the action is to wait) before time t_k. The rationale is that the maximum cost of a match in the matching formed so far does not change and thus we aim to compensate the negative rewards due to the consecutive actions of waiting. In the other sub-case that $c_{k+1} > c_k$, we define R_k to be $(c_k - c_{k+1}) + cnt$, where $(c_k - c_{k+1})$ is a negative reward since the maximum cost becomes worse after the action a_k is taken and cnt is used for compensation again. In summary, we have

(3.7)
$$R_k = \begin{cases} -1 & \text{if } \theta(B_k) < l \\ cnt & \text{if } \theta(B_k) \geq l \text{ and } c_{k+1} = c_k \\ (c_k - c_{k+1}) + cnt & \text{if } \theta(B_k) \geq l \text{ and } c_{k+1} > c_k \end{cases}$$

We would like to emphasize the trickiness of (1) defining the reward of a "wait" action to -1 and (2) compensating it with the rewards of "match and hold" actions. Normally, one may think when a "wait" action is taken, the reward should be defined to be 0 since the matching is not changed. But this definition would suffer from a few issues: (1) it fails to capture the semantics of waiting and outputs no negative signal for waiting though it would essentially increase the costs of matches; (2) in the case that we take a "match and hold" action and the objective is not changed, the reward would still be 0, resulting very sparse rewards (mostly zeros) and a model harder to train. These issues will be well solved with the definition in Equation 3.7.

Note that with the rewards defined as above, the goal of the MDP, i.e., to maximize the accumulative rewards, is aligned with that of the OBM-D problem. To see this, suppose that we go through a sequence of states $s_1, s_2, ..., s_N$ and correspondingly, we receive a sequence of rewards $R_1, R_2, ..., R_{N-1}$. In the case that the future rewards are not discounted, we have

(3.8)
$$\sum_{t=1}^{N-1} R_t = c_1 - c_N$$

where c_N is the maximum cost of a match after the last action is performed, which corresponds to the maximum cost of the matching formed.

Algorithm 1 The Adaptive-H algorithm

Require: A set R (resp. W) of n requests (resp. m workers) which arrive dynamically ($n \leq m$);
Ensure: A set of n matches which are formed sequentially;

for each time step t_k where k is 1, 2, ... **do**
  Batch $B_t \leftarrow$ the set containing all requests and workers that have arrived by t_k and not matched yet;
  State $s_t \leftarrow$ the state as constructed based on B_k (Refer to Section 3.1.1);
  Action $a_t \leftarrow \arg\max_a Q(s_t, a)$;
  if $a_k = l \leq \theta(B_t)$ **then**
    Find the bottleneck matching M within batch B_k;
    for each (r_i, w_j) in M **do**
      if $r_i.t \leq (t_k - l)$ **then**
        Output (r_i, w_j, t_k);
      end if
    end for
  end if
end for

3.2 Policy Learning on the MDP The core problem of a MDP is to find an optimal *policy* for the agent, which corresponds to a function that specifies the action that the agent should choose in a specific state so as to maximize the accumulative rewards. We learn the policy on the MDP constructed for the OBM-D problem via a *value-based* method called *Q-learning* [44]. The main idea is as follows. First, it defines an action-value function $Q(s, a)$ (or Q function), which represents the maximum amount of expected accumulative rewards we would receive by following any policy after seeing state s and taking action a. Second, it estimates $Q(s, a)$'s by using the following Bellman Equation [33] as an iterative update step:

(3.9)
$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [R + \gamma \cdot \max_{a'} Q^*(s', a') - Q(s, a)]$$

where s is the current state, a is the action that is chosen based on some policy derived from Q (e.g., using the ϵ-Greedy strategy), R and s' are the reward and the next state observed, respectively, α is the learning rate, and γ is a discount factor. Third, it returns the policy which always chooses the action that maximizes the $Q(s, a)$ function for a given state s.

Table 1: Statistics of Real Datasets.

Dataset	# of requests/workers	City, Country
Didi [6]	1,992,683	Haikou, China
Olist [7]	96,475	Sao Paulo, Brazil

3.3 The Adaptive-H Algorithm Our Adaptive-H algorithm, presented in Algorithm 1, is based on the learned policy for the MDP that models the OBM-D problem. Specifically, it proceeds iteratively at each time step t_k for $k = 1, 2, ...$ (Line 1). At t_k, it constructs a batch B_k, which involves all those requests and workers that have arrived before time t_k and not yet matched (Line 2). Based on B_k, it then constructs a state $s_t = (\theta(B_k), \sigma(B_k))$ (Refer to Section 3.1.1) (Line 3). Then, it takes an action a_k, which maximizes the Q function at state s_k, i.e., $a_k = \arg\max_a Q(s_t, a)$ (Line 4). Suppose $a_k = l$. It means to wait till the next time step t_{k+1} if the time span of B_k is smaller than l, i.e., $l > \theta(B_k)$, and to form a bottleneck matching within batch B_k otherwise (Lines 5-6). Furthermore, in the latter case, for each pair of r_i and w_j that are matched to each other in the bottleneck matching, it would return a match (r_i, w_j, t_k) if r_i arrived earlier than or at time $(t_k - l)$ and hold the matches among the remaining requests and workers (Lines 7-9).

The time complexity is dominated by the part of performing a bottleneck matching at some time steps, which is further determined by two factors: (1) the number of time steps at which the action involves performing a bottleneck matching; and (2) the sizes of the batches on which a bottleneck matching is performed. By the nature of the algorithm, these two factors cannot be large simultaneously, i.e., if the number of time steps is large, then the sizes of the batches are small. As will be shown in the empirical study, our Adaptive-H algorithm runs consistently faster than existing algorithms.

4 Experiments

4.1 Experimental Setup As shown in Table 1, we use two real datasets, namely *Didi* and *Olist*. The *Didi* dataset is collected from Didi Chuxing [2], which is the data of 1,992,683 taxi orders at Haikou, China, from 2017-05-01 to 2017-05-31 and published through its GAIA initiative [6]. Each taxi order contains six elements, namely pick-up latitude/longitude/timestamp and drop-off latitude/longitude/timestamp. Following existing studies [15, 42], for each taxi order, we create a request with its arriving time set to be the taxi order's pick-up time and also a worker with its arriving time set to be the taxi order's drop-off time (since at the drop-off time, a taxi, which corresponds to a worker, would become available to serve other requests). The *Olist* dataset is published at Kaggle [7], which contains real-world e-commerce orders of a Brazilian company called Olist [8]. Similar to [45], we only consider the orders with the customer's and seller's locations both at the city of Sao Paulo, Brazil, and collect 96,475 such orders from 2016 to 2018. For each order, we generate a request with its arrival time as the order's placement time and also a worker with its arrival time as the order's delivery time. Following [42], for each pair of a request r_i and a worker w_j in these datasets, we approximate $t(r_i, w_j)$ as the geodesic distance between r_i and w_j divided by a context-dependent speed (40km/h for *Didi* and 10km/h for *Olist*).

We also generate some synthetic datasets by following the existing study [15] and conduct experiments on the datasets by varying (1) cardinality n, (2) t_{max}, and (3) spatial and temporal distributions. The detailed results could be found in a technical report [41], which show similar clues as those on the real datasets.

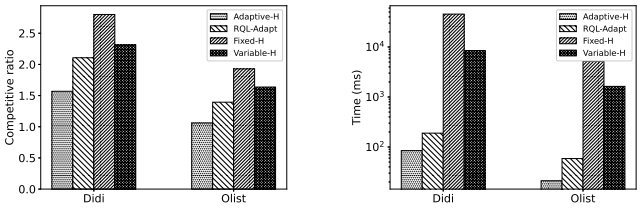
Metrics and Baselines. We compare our algorithm Adaptive-H with three existing algorithms, namely *Fixed-H* [30], *Variable-H* [15], and *RQL-Adapt* [42], in terms of the maximum cost and running time. Fixed-H devises a strategy, which always holds those matches with the costs among the bottom-k and matches the others, where k is a pre-specified parameter. Variable-H is the state-of-the-art algorithm, which is to hold *each* match for a certain amount of time that is dependent on the match. RQL is a RL based algorithm for the online bipartite matching problem with the objective of maximizing the *overall* matching utility/weight. In RQL, a state is defined as a pair of two integers, one equal to the number of requests and the other equal to the number of workers in the current batch, and an action is defined as a binary number, 0 denoting to wait and 1 denoting to match (i.e., this action definition provides no flexibility of holding matches). Since it targets a different objective, directly comparing our algorithm and RQL in terms of the objective of OBM-D is not fair. Therefore, we adapt RQL by replacing its reward definition with that of Adaptive-H so as to align its objective with that of OBM-D. We keep all other parts of RQL unchanged and call the adapted version *RQL-Adapt*.

Hyperparameter Selection and Model Training. The Adaptive-H method involves three hyperparameters: (1) the gap C between two adjacent time steps; (2) the bin size W for discretizing the state space; and (3) the size of the action space D. Fixed-H involves one hyperparameter, k, and RQL-Adapt involves two hyperparameters, namely $l_{\min}$ and $l_{\max}$, denoting the minimum and maximum number of objects in a batch for matching, respectively. We use *grid search* for finding the best configuration of hyperparameters for each algorithm and on each dataset. We report the detailed hyperparameter settings of different algorithms on different datasets in a technical report [41].

For the *Didi* dataset, which consists of data spanning over about 4 weeks, we use the data of the first three weeks for training and the data of the last week for testing. For the *Olist* dataset, which consists of data from 2016 to 2018, we use the data of 2016 and 2017 for training and that of 2018 for testing. During the training process, we employ

the ϵ-greedy strategy for selecting an action at a current state, i.e., with probability ϵ, we select a random action and with probability $(1-\epsilon)$, we select the action which maximizes the reward. The settings of the parameters that are involved in the training process include: (1) the number of episodes is set as 10,000; (2) the learning rate α is set as $1/(100+episode\ \ no.)$, i.e., the learning rate decreases episode after episode; (3) the ϵ in the ϵ-greedy procedure is set as 0.1 and (4) the discount rate γ is set as 1.

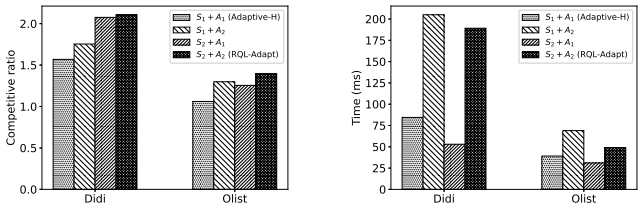
We implement all algorithms in C++. All experiments are conducted on a ubuntu machine with Intel(R) Core(TM) i5 2.70GHz CPU and 16GB main memory. Implementation code and datasets could be found via this link *https://github.com/wangkaixin219/OBM-D*.



(a) Max cost (b) Running time

Figure 2: Results on real datasets

4.2 Experimental Results (1) Results on Real Datasets. The results on real datasets are shown in Figure 2, where for the maximum costs, we show their competitive ratios. Consider the maximum cost (Figure 2(a)). Adaptive-H performs the best consistently among all online algorithms. Besides, RQL-Adapt performs better than Variable-H and Fixed-H. These results meet our expectation that an adaptive holding strategy works better than non-adaptive ones. In addition, Adaptive-H captures the OBM-D problem better than RQL-Adapt (detailed elaborations into this shall be provided in the ablation study part). Consider the running time (Figure 2(b)). Adaptive-H runs faster than all existing algorithms. Explaining these results by comparing the worst-case time complexities is not straightforward, since the time complexity captures the cost of matching and holding for each step, but the practical efficiency relies on how frequently these actions are performed. Therefore, we investigated some running statistics of the algorithms for explanations. Take the results on the *Didi* dataset as an example. For Adaptive-H, it totally takes 17,323 actions, among which 14,337 actions are to wait while only 2,986 actions are to match and hold (on average), i.e., the percentage of the actions of waiting (which are cheap) is rather high, e.g., about 82.7%. For *RQL-Adapt*, however, it takes 17,321 actions in all, among which 6,925 actions are to match, which is significantly larger than that for Adaptive-H, i.e., 2,986. For Fixed-H, under the same settings, it performs matching at 8,492 time steps, which is also much larger than 2,986. For Variable-H, while it involves no procedure of matching since it matches workers greedily, it involves many cases where a match is formed first and then broken because a better match is possible. In this case, a worker is considered as a "new" worker to be matched quite a few times, causing extra workload.



(a) Max cost (b) Running time

Figure 3: Results of ablation study

(2) Ablation Study (Real Datasets). In order to evaluate the effectiveness of the state and action definitions of Adaptive-H, separately and collectively, we replace each of them or both with some alternative definitions, namely those adopted in RQL [42]. We denote the state and action definitions of Adaptive-H (RQL) by S_1 and A_1 (S_2 and A_2), respectively. In this experiment, we explore four combinations of state and action definitions, namely, (1) S_1+A_1 (i.e., Adaptive-H), (2) S_1+A_2, (3) S_2+A_1, and (4) S_2+A_2 (i.e., RQL or RQL-Adapt). The results on the real datasets are shown in Figure 3, where for the maximum costs, we show their corresponding competitive ratios. Consider the effectiveness (Figure 3(a)). We observe that S_1+A_1 (i.e., Adaptive-H) performs the best and S_2+A_2 (i.e., RQL-Adapt) performs the worst, which could be possibly explained by that a state of Adaptive-H captures richer and more relevant information and an action makes it possible to perform holding. Consider the efficiency (Figure 3(b)). We observe that S_2+A_1 runs the fastest, which could be explained by that (1) S_2 is cheaper to compute than S_1 and (2) with A_1, it possibly makes less yet high-quality match operations. In addition, S_1+A_1 (Adaptive-H) runs the second fastest, only slightly slower than S_2+A_1.

5 Related Work

Bottleneck Matching Problems. The bottleneck matching problem, also known as min-max assignment or worst case matching problem, was first proposed in [22]. The offline version of the problem was studied in [14, 21, 32], where the objects to be matched are assumed to be available at the very beginning. The one-sided online version of bottleneck matching was formally studied in [25] and [29], where one set of objects is given beforehand and the problem is to assign a set of servers to a set of tasks with the same size such that the maximum distance between a server and a task is minimized. In [24], the authors presented a lower bound of competitive ratio of any deterministic algorithm for the online bottleneck matching, which is $\frac{n}{\ln 2}$ and in [13], the authors presented one of any randomized algorithm

for the one-sided online bottleneck matching, which is $\frac{n}{2}$. In [9], the authors perform resource augmentation analysis to examine the performance of algorithms and conclude that the competitive ratio still remains linear when an extra server is introduced at each server-vertex. There exist some studies [20, 11, 10, 12, 15, 30], where the cost of a match depends on when it is formed, i.e., it is delay-aware, and among them, only [15, 30] adopt a delay-aware matching objective as ours. Specifically, in [15], the OBM-D problem was first proposed and in [30], a variant of OBM-D was studied. Nevertheless, as explained in Section 1, the holding strategies used in these studies cannot be adaptive to the problem dynamics and these studies provide no analysis on the competitive ratios of randomized algorithms for OBM-D.

Reinforcement Learning. Reinforcement learning assumes that agents are in an unknown environment, seeking to achieve a goal, in which agents need to learn how to map situations to actions so as to maximize a numerical reward signal. The environment is typically formulated as a Markov Decision Process (MDP) [34]. In this paper, we adopt the Q-learning algorithm [44] for the MDP that models the OBM-D problem since it does not require a model of the environment, has the ability to handle problems with stochastic transitions and rewards, and has guarantees on convergence. While some existing studies have attempted to use reinforcement learning for matching problems [18, 36, 42, 28, 43, 31, 23, 35], they differ from the study in this paper in (1) they target offline settings [18]; (2) they target max-sum/min-sum objectives [28, 36, 42], which result in very different design strategies from ours; and (3) they target some specialized matching problems such as order dispatching [43, 31, 23] and task assignment [35], which assume richer domain contexts (e.g., spatial information) that we do not assume available in this paper and aim to optimize domain-related objectives (e.g., revenues) that are different from ours. For example, the holding strategy is not touched in any of these methods, but is critical for solving a bottleneck matching problem that is targeted in this paper.

6 Conclusion

In this paper, we study the online bottleneck matching with delays (OBM-D) problem, for which, a holding strategy that holds a match for a while when it is first formed is critical. Nevertheless, existing holding strategies are all based on some human-crafted rules. In this paper, we propose a new holding strategy which is adaptive and based on reinforcement learning, and develop a method named Adaptive-H for OBM-D. In addition, we provide theoretical results on the performance boundary that a randomized algorithm that could achieve for OBM-D, which are the first of their kind for OBM-D. We conduct extensive experiments on both real and synthetic datasets, showing that our Adaptive-H algorithm outperforms existing algorithms in terms of both efficiency and effectiveness. An interesting future research direction is to explore adaptive algorithms for the online matching problem on general graphs.

Acknowledgments. We thank the anonymous reviewers for their valuable suggestions and comments. This research is supported by the Nanyang Technological University Start-UP Grant from the College of Engineering under Grant M4082302 and by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG20/19 (S)). Yongxin Tong and Yi Xu's work was partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant No. 61822201 and U1811463. This work was also partially supported by the A*STAR Cyber-Physical Production System (CPPS) – Towards Contextual and Intelligent Response Research Program, under the RIE2020 IAF-PP Grant A19C1a0018, and Model Factory@SIMTech.

References

[1] Uber. *https://www.uber.com*.
[2] Didi chuxing. *https://www.didichuxing.com*.
[3] Grab. *https://www.grab.com*.
[4] Ele.me. *https://www.ele.me*.
[5] Grubhub. *https://www.grubhub.com*.
[6] Gaia. *https://outreach.didichuxing.com/research/opendata/*.
[7] Olist dataset published in Kaggle. *https://www.kaggle.com/olistbr/brazilian-ecommerce*.
[8] Olist. *https://www.crunchbase.com/organization/olist*.
[9] B. M. Anthony and C. Chung, *Online bottleneck matching*, Journal of Combinatorial Optimization, 27 (2014), pp. 100–114.
[10] I. Ashlagi, Y. Azar, M. Charikar, A. Chiplunkar, O. Geri, H. Kaplan, R. Makhijani, Y. Wang, and R. Wattenhofer, *Min-cost bipartite perfect matching with delays*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[11] I. Ashlagi, Y. Azar, M. Charikar, A. Chiplunkar, O. Geri, H. Kaplan, R. Makhijani, Y. Wang, and R. Wattenhofer, *Min-cost matching with delays*, in 20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), Berkeley, California, USA.(August 2017), 2017.
[12] Y. Azar, A. Chiplunkar, and H. Kaplan, *Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays*, in Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2017, pp. 1051–1061.
[13] M. C. W. Bender and C. Chung, *Lowerbounds for the online minimum matching problem on the line*, in Mit Undergraduate Research Technology Conference, 2016.
[14] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*, Springer, 2009.

[15] Z. Chen, P. Cheng, Y. Zeng, and L. Chen, *Minimizing maximum delay of task assignment in spatial crowdsourcing*, in 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 1454–1465.

[16] P. Cheng, X. Lian, L. Chen, and C. Shahabi, *Prediction-based task assignment in spatial crowdsourcing*, in 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 997–1008.

[17] P. Cheng, X. Lian, Z. Chen, R. Fu, L. Chen, J. Han, and J. Zhao, *Reliable diversity-based spatial crowdsourcing by moving workers*, Proceedings of the VLDB Endowment, 8 (2015), pp. 1022–1033.

[18] E. Curry et al., *Adaptive task assignment in spatial crowdsourcing*, PhD thesis, 2016.

[19] J. Dickerson, K. Sankararaman, A. Srinivasan, and P. Xu, *Allocation problems in ride-sharing platforms: Online matching with offline reusable resources*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[20] Y. Emek, S. Kutten, and R. Wattenhofer, *Online matching: haste makes waste!*, in Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, ACM, 2016, pp. 333–344.

[21] H. N. Gabow and R. E. Tarjan, *Algorithms for two bottleneck optimization problems*, Journal of Algorithms, 9 (1988), pp. 411–417.

[22] O. Gross, *The bottleneck assignment problem*, tech. report, RAND CORP SANTA MONICA CALIF, 1959.

[23] J. Holler, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye, *Deep q-learning approaches to dynamic multi-driver dispatching and repositioning*, in NeurIPS 2018 Deep Reinforcement Learning Workshop, 2018.

[24] R. Idury and A. Schaffer, *A better lower bound for online bottleneck matching*. http://www.ncbi.nlm.nih.gov/core/assets/cbb/files/Firehouse.pdf, 1992.

[25] B. Kalyanasundaram and K. Pruhs, *Online weighted matching*, Journal of Algorithms, 14 (1993), pp. 478–488.

[26] L. Kazemi and C. Shahabi, *Geocrowd: enabling query answering with spatial crowdsourcing*, in Proceedings of the 20th international conference on advances in geographic information systems, ACM, 2012, pp. 189–198.

[27] L. Kazemi, C. Shahabi, and L. Chen, *Geotrucrowd: trustworthy query answering with spatial crowdsourcing*, in Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems, ACM, 2013, pp. 314–323.

[28] J. Ke, F. Xiao, H. Yang, and J. Ye, *Optimizing online matching for ride-sourcing services with multi-agent deep reinforcement learning*, arXiv preprint arXiv:1902.06228, (2019).

[29] S. Khuller, S. G. Mitchell, and V. V. Vazirani, *On-line algorithms for weighted bipartite matching and stable marriages*, Theoretical Computer Science, 127 (1994), pp. 255–267.

[30] L. Li, J. Fang, B. Du, and W. Lv, *Spatial bottleneck minimum task assignment with time-delay*, in International Conference on Database Systems for Advanced Applications, Springer, 2019, pp. 387–391.

[31] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, *Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning*, in The World Wide Web Conference, 2019, pp. 983–994.

[32] C. Long, R. C.-W. Wong, P. S. Yu, and M. Jiang, *On optimal worst-case matching*, in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 845–856.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), p. 529.

[34] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.

[35] C. Shan, N. Mamoulis, R. Cheng, G. Li, X. Li, and Y. Qian, *An end-to-end deep rl framework for task arrangement in crowdsourcing platforms*, in 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, 2020, pp. 49–60.

[36] M. Z. Spivey and W. B. Powell, *The dynamic assignment problem*, Transportation Science, 38 (2004), pp. 399–419.

[37] H. To, C. Shahabi, and L. Kazemi, *A server-assigned spatial crowdsourcing framework*, ACM Transactions on Spatial Algorithms and Systems, 1 (2015), p. 2.

[38] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, *Online minimum matching in real-time spatial data: experiments and analysis*, Proceedings of the VLDB Endowment, 9 (2016), pp. 1053–1064.

[39] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, *Online mobile micro-task allocation in spatial crowdsourcing*, in 2016 IEEE 32Nd international conference on data engineering (ICDE), IEEE, 2016, pp. 49–60.

[40] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, *Flexible online task assignment in real-time spatial data*, Proceedings of the VLDB Endowment, 10 (2017), pp. 1334–1345.

[41] K. Wang, C. Long, Y. Tong, J. Zhang, and Y. Xu, *Adaptive holding for online bottleneck matching with delays (technical report)*. https://personal.ntu.edu.sg/c.long/paper/bottleneck-technical.pdf, 2021.

[42] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, *Adaptive dynamic bipartite graph matching: A reinforcement learning approach*, in 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 1478–1489.

[43] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu, *Deep reinforcement learning with knowledge transfer for online rides order dispatching*, in 2018 IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 617–626.

[44] C. J. Watkins and P. Dayan, *Q-learning*, Machine learning, 8 (1992), pp. 279–292.

[45] Y. Zeng, Y. Tong, and L. Chen, *Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees*, Proceedings of the VLDB Endowment, 13 (2019), pp. 320–333.

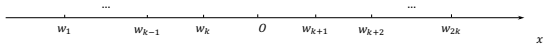
Adaptive Holding for Online Bottleneck Matching with Delays (Supplementary Materials)

Kaixin Wang* Cheng Long* Yongxin Tong† Jie Zhang* Yi Xu†

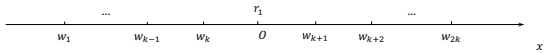
1 Proof of Theorem 2.1

Proof. We prove by constructing an instance with $n = 2k$ workers w_{1-2k} and requests r_{1-2k} located on an axis. We assume that the cost between a worker and a request is equal to the distance between their locations. Specifically, the workers and requests arrive as follows, which are shown in Figure 1.

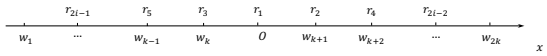
1. At time $t_0 = 0$, workers w_1 to w_{2k} arrive at $-k, -k+1, ..., -1, 1, ..., k-1, k$, respectively.
2. At time $t_1 = 1$, request r_1 arrives at $l_{r_1} = 0$;
3. At time $t_i = i$, r_{2i-2} arrives at $l_{r_{2i-2}} = i - 1$ and r_{2i-1} arrives at $l_{r_{2i-1}} = -i + 1$ for $i = 2, 3, \cdots, k$. After the first $2k - 1$ requests arrive, we wait until $t_{k+1} = 2k$.
4. At time $t_{k+1} = 2k$, request r_{2k} arrives at either k or $-k$ with equal probabilities, which yields a probability distribution X over the input of requests.



(a) At time $t_0 = 0$, all workers arrive.



(b) At time $t_1 = 1$, request r_1 arrives.



(c) At time $t_i = i$, requests r_{2i-2} and r_{2i-1} arrive.

Figure 1: Arriving dynamics of workers and requests

The objective of the offline optimal solution would be equal to 1, which is explained as follows. In the case that request r_{2k} arrives at k, then any request on the positive side can be assigned to a certain worker whose location is the same as the request's while any request on the negative side can be assigned to a certain worker whose location is one unit left to the request. In the case that request r_{2k} arrives at $-k$, then any request on the positive side can be assigned to a certain worker whose location is one unit right to the request while any request on the negative side could be assigned to a certain worker whose location is the same as the request's.

For any online deterministic algorithms, they can be classified into two types according to the way how it assigns the first $2k - 1$ requests, denoted by $\mathcal{A}_1$ and $\mathcal{A}_2$.

The first set of algorithms $\mathcal{A}_1$ is to match some of the first $2k - 1$ workers after r_{2k}'s arrival. Then, there exists at least one request whose waiting time is greater or equal than k. Then the maximum of all requests' waiting time is no less than k. Thus, for the first type, the competitive ratio is no less than $\frac{k}{1} = k = \frac{n}{2}$.

The other type $\mathcal{A}_2$ is to match all the first $2k - 1$ requests before the arrival of r_{2k}. Note that after r_{2k-1} has been matched, there would always be exactly one worker remaining, and we denote its location by $s \in \mathbb{Z}$. Because request r_{2k} would be at k or $-k$ with equal probabilities, the expectation of the cost value under the input distribution X is at least

$$\min_{A \in \mathcal{A}_2} \mathbb{E}\big[obj(A(X))\big] = \frac{1}{2}(s + k) + \frac{1}{2}(k - s) = k$$

Since no deterministic online algorithm can achieve a result better than $\frac{n}{2}$ under the distribution X, by using Yao's principle [2], for any randomized algorithm, we have

$$\max_{x \in \mathcal{X}} \mathbb{E}\big[obj(A(x))\big] \geqslant \min_{A \in \mathcal{A}_2} \mathbb{E}\big[obj(A(X))\big] = k = \frac{n}{2}$$

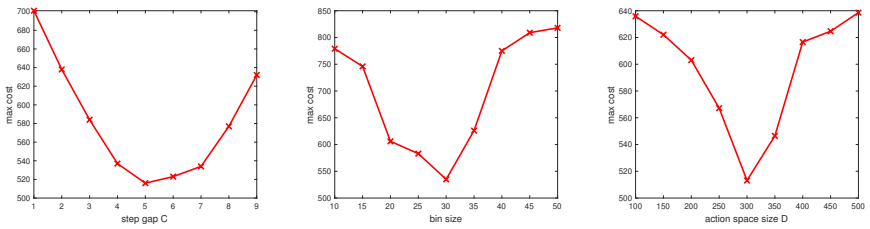
Therefore, we conclude that no randomized algorithm for the OBM-D problem with n workers (requests) can achieve a competitive ratio better than $\frac{n}{2}$. □

2 Additional Experiments and Results

Synthetic Datasets. We follow the existing study [1] for generating synthetic datasets. Specifically, we assume that both requests and workers are located in the grid cells of a 2D spatial space, and the cost between a request and a worker is defined as the number of grid cells the worker needs to traverse in order to reach the grid cell in which the request is

*Nanyang Technological University, Singapore, {kaixin.wang, c.long, zhangj}@ntu.edu.sg

†Beihang University, China, {yxtong, xuy}@buaa.edu.cn

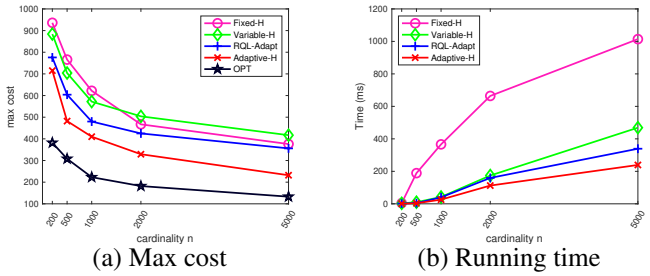


(a) max cost of varying step gap (b) max cost of varying bin size (c) max cost of varying action space size

Figure 2: Results on varying time step gap C, bin size W and action space size D on synthetic dataset

located. We explore two distributions, namely Uniform and Gaussian, with appropriate parameters for generating the locations (or grid cells) of requests and workers, and use the Uniform distribution as the default one. In addition, we assume that the same number n of requests and workers arrive dynamically following some distribution during a predefined time window $(0, 1, 2, ..., t_{max})$. We explore three distributions, namely Uniform, Gaussian, and Zipf, with appropriate distribution parameters for the arriving times, and use the Uniform distribution as the default one. For experiments on synthetic datasets, given a setting, the datasets are generated 20 times and the average results are reported.

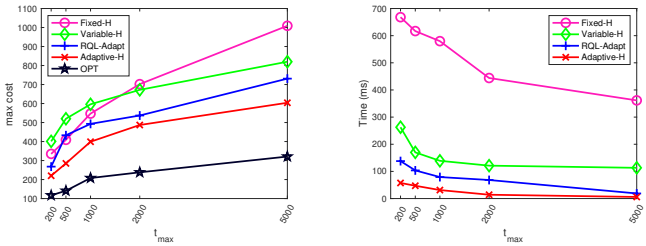
Hyperparameter Selection. In Figure 2, we show the results of the performance of our method when varying one of the hyperparameters while fixing the other two with their default settings under the synthetic dataset. On the synthetic dataset, our method has the best performance under the settings of $C = 5$, $W = 30$, and $D = 300$. On the real datasets Didi and Olist, we also adopt the same process for tuning the hyperparameters, and the results show the same trends. We have included the best hyperparameter setting for each real dataset with our codes at *https://anonymous.4open.science/r/0466e3c1-7381-410e-bc53-400bd9b1ed5f/*.



(a) Max cost (b) Running time

Figure 3: Results of varying cardinality (synthetic datasets)

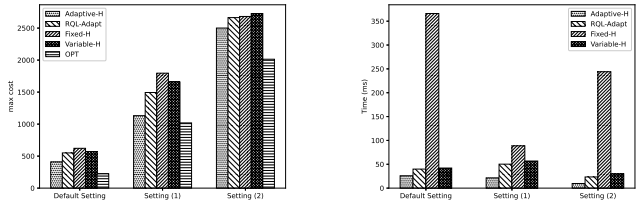
(1) Effects of Cardinality n (Synthetic Datasets). The results of varying the cardinality n, i.e., the number of requests/workers, are presented in Figure 3, where $t_{\max} = 2000$. Consider the maximum cost (Figure 3(a)). We observe that it decreases when n increases. This could be explained by the fact that with more requests and workers, the density would be higher and in general it would take less time for a worker to travel to a request. Besides, our Adaptive-H algorithm outperforms other algorithms over all settings. Consider the running time (Figure 3(b)). All algorithms have their running times increase when n increases. Besides, Adaptive-H runs faster than other algorithms over all settings.



(a) Max cost (b) Running time

Figure 4: Results of varying $t_{\max}$ (synthetic datasets)

(2) Effects of $t_{\max}$ (Synthetic Datasets). The results of varying $t_{\max}$, i.e., the maximum time steps, are presented in Figure 4, where $n = 1000$. All algorithms return smaller maximum costs when $t_{\max}$ increases. This could be explained by the fact that when $t_{\max}$ gets larger, which implies that the requests and workers arrive within a longer period of time, the requests would usually need to wait for a longer time before a favorable worker could be matched to it, resulting a larger cost. Besides, Adaptive-H outperforms other algorithms in terms of maximum cost and running time over all settings.



(a) Max cost (b) Running time

Figure 5: Results of varying distributions

(3) Effects of Distribution Settings (Synthetic Datasets). We denote the distributions Uniform (location), Gaussian

$N(\mu_s = 500, \sigma_s = 50)$ (location), Uniform (arriving time), Gaussian $N(\mu_t = \frac{t_{max}}{2}, \sigma_t = 200)$ (arriving time), and Zipf $f(N = t_{\max}, s = 2)$ (arriving time) by L_1, L_2, T_1, T_2, and T_3, respectively. We follow [1] and test two specific distribution settings (apart from the default one, i.e., (L_1, T_1) for both requests and workers): (1) (L_2, T_2) for requests and (L_1, T_1) for workers; and (2) (L_1, T_3) for requests and (L_1, T_1) for workers. The results are shown in Figure 5. We observe that Adaptive-H enjoys its advantages of effectiveness and efficiency across different distribution settings. In addition, consider the Setting (1). The maximum costs become larger compared with those under the default distribution setting. This could be explained as follows. First, since requests' locations follow the Gaussian distribution, they tend to arrive near the center. For those workers whose locations are far from the center, they would need to travel a long distance. Second, since the requests' arrival times follow the Gaussian distribution, the requests tend to arrive at around $\frac{t_{\max}}{2}$. Some workers would arrive late and the waiting times of requests would be large. Consider the Setting (2). Since the requests' arrival times follow the zipf distribution, up to 80% of requests arrive early, whereas workers' arrival times follow a Uniform distribution. As a result, the maximum costs under this distribution setting become even larger. We also notice that Fixed-H runs exceptionally slow for the default setting and Setting (2). This is possibly because for these settings, the requests and workers arrive in a more scattered way and Fixed-H would perform more matchings.

References

[1] Z. CHEN, P. CHENG, Y. ZENG, AND L. CHEN, *Minimizing maximum delay of task assignment in spatial crowdsourcing*, in 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, 2019, pp. 1454–1465.

[2] A. C.-C. YAO, *Probabilistic computations: Toward a unified measure of complexity*, in 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 222–227.