IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488692

Dynamically Choosing the Candidate Algorithm with Ostasos in Online Optimization

Weirong Chen†, Jiaqi Zheng†, Haoyu Yu†
†*State Key Laboratory for Novel Software Technology, Nanjing University, China*

***Abstract*—The increasing challenge in designing online algorithms lies in the distribution uncertainty. To cope with the distribution variations in online optimization, an intuitive idea is to reselect an algorithm from the candidate set that will be more suitable to future distributions. In this paper, we propose Ostasos, an automatic algorithm selection framework that can choose the most suitable algorithm on the fly with provable guarantees. Rigorous theoretical analysis demonstrates that the performance of Ostasos is no worse than that of any candidate algorithms in terms of *competitive ratio*. Finally, we apply Ostasos to the online car-hailing problem and trace-driven experiments verify the effectiveness of Ostasos.**

***Index Terms*—Online optimization, online learning, online bipartite matching, deep reinforcement learning**

I. Introduction

Online optimization are ubiquitous in many research fields such as prediction from expert advice [1], portfolio selection [2], recommendation systems [3], task assignment [4], *etc.* Instead of offline optimization, online optimization problems need to be solved piece-by-piece in a serial fashion with limited input information and iteratively make decisions. A loss may be produced after the decision is committed and our goal is to minimize the total loss in the long run.

Adversary and independent identically distributed (*i.i.d.*) are two major settings when analyzing online algorithms. In the former, all input information are given by an adversary who deliberately degrades the performance of the online algorithm. In the latter, all inputs are sampled from a known or unknown distribution. However, in practice, the adversary settings are too pessimistic to navigate the algorithm design space. The distribution may not be fixed and it can always vary with time. Hence, we need to design more powerful online algorithms that can well respond to the potential distribution variations.

Reinforcement learning adapts to dynamic distributions by learning from the environment restlessly [5], [6], especially for the inputs with Markov property. However, this usually leads to inefficient repetitive learning especially when the distribution changes at first and recovers later. The repetitive learning procedure can degrade the performance and should be avoided. Online learning overcome this issue by restarting the algorithm periodically [7], [8] or "forgetting" old information gradually [9]. A lot of online algorithms with theoretical guarantees have been proposed to tackle dynamic distributions [10]–[12]. But due to the limitation of its simple model, (*i.e.*, it only exploits the reward of each history action and cannot explore the influence of inter-actions.) it cannot work well in complex problems with Markov property. Therefore, it remains an open problem whether there exists a better online algorithm that can adapt to the distribution variations.

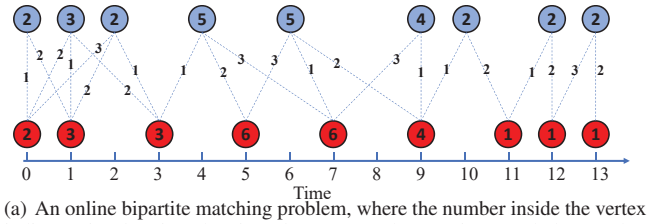
An intuitive idea is to reselect a more suitable algorithm after the distribution changes. However, none of the existing work gives the theoretical guarantee analysis in terms of competitive ratio. In this paper, we initially give a positive answer to the fundamental problem whether we can dynamically select different candidate algorithms to solve one online optimization problem with performance guarantees. To achieve performance guarantees, which conditions the candidate algorithm set needs to meet and how to dynamically select the most suitable candidate one on the fly are two core challenges. For the first challenge, we derive one necessary condition: the candidate algorithm set needs including at least one algorithm with provable competitive ratio. For the second challenge, we propose a dynamic restarting policy to re-initialize the selector algorithm once the distribution changes so that the most suitable algorithm can be selected from the candidate set.

Our first contribution is that we propose Ostasos — a universal algorithm selection framework in response to dynamic distributions. Ostasos has three components: a candidate algorithm set that can tackle a specific online optimization problem, a selector algorithm that can reselect the most suitable one from the candidate algorithm set, and a dynamic restarting policy used to detect the distribution variations. Once the distribution changes are detected, we restart the selector algorithm. We prove that, in terms of competitive analysis, Ostasos can select the fixed single best algorithm.

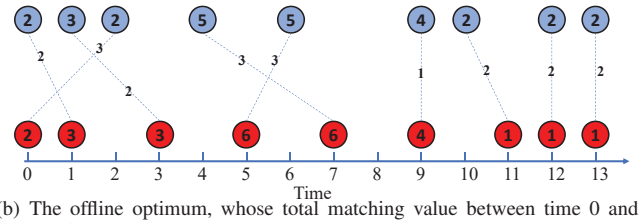
Our second contribution is that we apply Ostasos to the online car-hailing problem. We generalize the Batch algorithm [13] to the condition that each vertices' duration is unknown and prove that they have provable *competitive ratio*. Furthermore, we improve the Batch algorithms with the deep Q-learning network [5] to produce a complete candidate algorithm set. We prove that the resulting candidate set satisfies the necessary condition and Ostasos has theoretical guarantee in the problem.

Our third contribution is that we conduct extensive evaluations. We evaluate Ostasos based on a real dataset in the online car-hailing scenario. The experiment results show that Ostasos achieves the same performance as the optimal single algorithms in different periods and outperforms other algorithm-selecting schemes. The high true positive rate and low false positive rate of the variation detections validate that the restarting policy can effectively detect the variations.

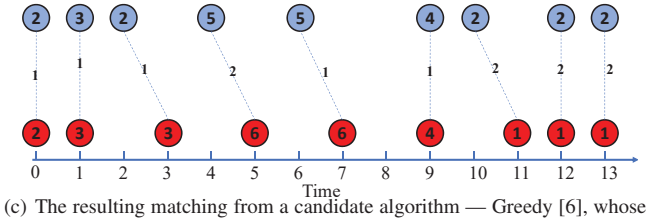
978-1-6654-0325-2/21/$31.00 ©2021 IEEE



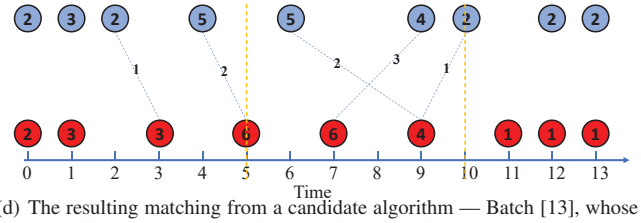
(a) An online bipartite matching problem, where the number inside the vertex indicates its duration that is unknown to other vertices, and the number associated with a edge indicates a potential matching value.



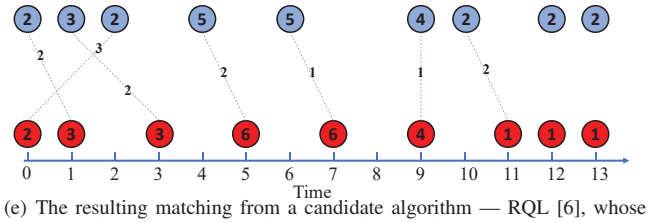
(b) The offline optimum, whose total matching value between time 0 and time 13 is 20.



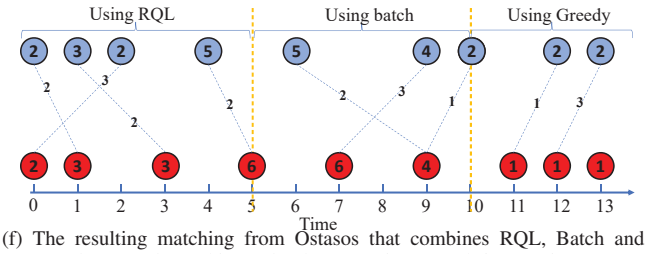
(c) The resulting matching from a candidate algorithm — Greedy [6], whose total matching value between time 0 and time 13 is 13.



(d) The resulting matching from a candidate algorithm — Batch [13], whose total matching value between time 0 and time 13 is 9.



(e) The resulting matching from a candidate algorithm — RQL [6], whose total matching value between time 0 and time 13 is 13.



(f) The resulting matching from Ostasos that combines RQL, Batch and Greey, whose total matching value between time 0 and time 13 is 19.

Fig. 1. An online bipartite matching problem and its possible solutions

II. A Motivating Example

We start with a motivating example to illustrate the potential benefits of Ostasos. Online bipartite matching problems can capture a large number of applications such as taxi-hailing, ride-sharing and kidney exchange. The vertices in each side of a bipartite graph arrive and depart over time in an online manner. Each vertices pair can produce a different matching value and the objective is to maximize the total matching value over a finite time horizon. Fig. 1(a) use different colors to represent the vertices in each side, in which the number inside the vertex indicates its duration and the number associated with a edge indicates a potential matching value.

The difficulty lies in that the duration is unknown and we cannot determine whether there would be another more suitable vertex for the current matching or not. In Fig. 1(a), the vertices arrive with unknown duration that makes us difficult to determine whether we should match the current vertex immediately or wait for a better choice. Specifically at time 0, two vertices colored blue and red arrive simultaneously. At this point, if these two vertices were matched together, we would obtain one unit matching value. But actually we would obtain a higher matching value if the vertices were matched as shown in Fig. 1(b). The offline optimum assumes that the vertices arriving order and their duration are both known in advanced, which cannot hold in practice. Yet it can be as a benchmark to measure the performance of other online algorithms.

Existing work especially Greedy, RQL and Batch can address the online bipartite matching problem. All vertices arrive and depart over the time horizon, which forms a dynamic bipartite graph. Greedy [6] is an intuitive policy and solve this problem in a bipartite graph corresponding to current time point. Greedy will match all vertices in current bipartite graph directly using Hungarian algorithm [14] to obtain a local optimal solution, without future knowledge. The resulting matching is shown in Fig. 1(c). Essentially, Greedy only care about the current bipartite graph and don't wait for a potential better choice in the future. We can totally get 13 unit matching value when we use Greedy.

Different from Greedy, Batch [13] divides the time horizon into a sequence of fixed intervals and each newly arrived vertex cannot be matched until the end of this time interval. This means that some vertices may be not matched as their duration may be less than the time interval. At the end of the time interval, the matching in the bipartite graph can be obtained by the Hungarian algorithm. In Fig. 1(d), we set the batch size to be 5. At time 5, Batch matches all survival vertices in the bipartite graph. Obviously, the certain vertices have already departed. Similarly, at time 10 and time 15, we determine the matching among all left vertices in the bipartite graph. The total matching value is 9 using Batch when the batch size is 5. The results indicate that when the batch size is suitable for the duration, it can achieve a good result. On the contrary, when it isn't suitable for the duration, the performance is bad.

RQL [6] uses Q-learning approach to learn how to choose a best matching over the time horizon. Actually, it aims to learn an unknown distribution in nature by the reinforcement

learning approach. Hence, it inevitably needs some time to train the model. This may perform perfectly when the real input can well match the training model. However, it would perform very bad once the distributions change significantly. As shown in Fig. 1(e), RQL performs very well at the beginning and it performs bad after time 4 due to the varying distributions. The main drawback of RQL is that it costs large amount of time to learn a new distribution which may cause huge loss for a long time. The training overhead under different distributions could degrade the performance. We can obtain 13 unit matching value between time 0 and time 13 when we use RQL.

Instead of using one single candidate algorithm — Greedy, Batch or RQL to solve online bipartite matching problem, we argue that combing their advantages together will produce a better solution. As shown in Fig. 1(f), if we can use RQL or Batch algorithm with batch size 2 before time 5 and then switch to Batch algorithm with batch size 5 after time 5 until time 10, and use Greedy or Batch algorithm with batch size 1 after time 10, we can get very high total reward, 19, which is very close to the offline optimum, 20. However, the central challenge is which conditions the candidate algorithm set needs to meet and how to dynamically select the most suitable candidate one on the fly, which is our focus in the following sections.

III. An Optimization Framework

In this section, we propose Ostasos, that can dynamically select the most suitable candidate algorithm in response to the changing distributions.

A. Problem Statement and Preliminaries

We use $x_t \sim \mathcal{D}_t$ to capture the potential input sampled from an unknown independent identical distribution $\mathcal{D}_t$ for each $t \in T$. For an online optimization problem, a decision maker selects a candidate algorithm $A_t^i \in \mathcal{A}$ at the beginning of each epoch, and a reward R_{x_t,A_t} is given to the decision maker, where we use $\mathbf{R}_{x_t} = (R_{x_t,A^1}, R_{x_t,A^2}, ..., R_{x_t,A^{|\mathcal{A}|}})$ to denote the reward vector and let R_{x_t,A_t} to denote the reward of A_t in $\mathbf{R}_{x_t}$. For convenience, we also use R_{t,A_t} to denote the reward of A_t at epoch t. Since x_t is sampled from $\mathcal{D}_t$, we use vector $\boldsymbol{\mu}_t = (\mu_{t,A^1}, \mu_{t,A^2}, ..., \mu_{t,A^{|\mathcal{A}|}})$ to denote the expectation vector of reward at epoch t, in which $\mu_{t,A} = \sum_{x_t \sim \mathcal{D}_t} \Pr(x_t) R_{x_t,A}$ where $\Pr(x_t)$ is the probability of sampling x_t from $\mathcal{D}_t$. Hence, once the distributions of the future input changes, $\boldsymbol{\mu}_t$ changes. The objective is to maximize the total expectations of rewards, *i.e.*, $\max_{A_t} \sum_{t=0}^{T} \mu_{t,A_t}$.

We use our motivating example of online bipartite matching to explain the notations above. Greedy, Batch, and RQL are three candidate algorithms in set $\mathcal{A}$. When the vertex comes and when to leave out have a significant impact on the performance of online matching. In general, when the duration is relatively short, the vertex should be matched as soon as possible and Greedy performs better. On the contrary, if the vertex can wait for quite a long time, Batch or RQL tends to make a better decision. At epoch t, the arrived vertices, the vertices' duration, and the edges' weights constitute the input x_t. Since the duration is unknown, x_t is partially unknown. When we pick a specific algorithm A at epoch t, a reward $R_{x_t,A}$ with expectation $\mu_{t,A}$ will be incurred.

Here we use *competitive ratio* to measure the gap between Ostasos and the offline optimum, while *Regret* is used to measure the ability to select an optimal algorithm from $\mathcal{A}$ on the fly. Following [15] and [16], we define algorithm A's *competitive ratio* $CR(A) = \rho$. For any sequence of distributions $(\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_T)$, we have

$$\sum_{t=1}^{T} \mu_{t,A} \geq \rho \cdot \sum_{t}^{T} \mu_{t,OPT} - O(1) \tag{1}$$

where $\mu_{t,OPT}$ is the expectation of offline optimum at epoch t. Note that when A has no *competitive ratio* guarantee, we consider $\rho = 0$. We introduce *Static Regret*.

Static Regret [17] is the difference between the reward of actual choices and that of a single optimal choice A^* in all epoches,

$$Regret_S = \max_{A^*} \left(\sum_{t}^{T} \mu_{t,A^*} - \sum_{t}^{T} \mu_{t,A_t} \right)$$

where A_t is the selected candidate algorithm at each epoch t.

We present a common assumption, which is in line with that in prior work [18].

Assumption 1. *[18] All rewards $\{R_{x_t,A}\}$ in the reward vector have an upper bound and lower bound,* i.e.*,*

$$\forall\ \mathcal{D}_t,\ and\ A \in \mathcal{A},\ C \leq R_{x_t,A} \leq D$$

where C and D are constant and greater than zero.

B. Ostasos Design

Ostasos includes an universal framework — Algorithm 1 and a UCB-based selector algorithm — Algorithm 2. We first show the detailed procedure of Algorithm 1. At first, we initialize two fixed-size queues, Q_0 and Q_1, of which the sizes are l and $l/2$, respectively (line 1). At the beginning of every epoch t, a candidate algorithm $A \in \mathcal{A}$ is obtained by running Algorithm 2 based on history rewards (line 3). At the same time, Algorithm 2 obtains the reward. Q_0 records the algorithm-reward pair (A_t, R_{t,A_t}) in the latest l epochs (line 4) and the index of this pair is θ, where $Q_0(\theta)(0)$ and $Q_0(\theta)(1)$ denote the candidate algorithm and the reward respectively. When the length of Q_0 reaches l, Algorithm 1 first constructs two temporary reward vectors $\mathbf{R}'$ and $\mathbf{R}''$ (lines 6). And then Algorithm 1 determines the index θ_1 and θ_2. Property 1 can guarantee the existence of θ_1 and θ_2, which will be discussed later. Next $\mathbf{R}'$ and $\mathbf{R}''$ are assigned to $Q_0(\theta_1)(1)$ and $Q_0(\theta_2)(1)$ (lines 9-10). Note that if a candidate algorithm is stored more than once during $l/2$ epoches, the corresponding reward may not unique in the first half of Q_0. At this point, Algorithm 1 randomly selects one reward to construct $\mathbf{R}'$ or $\mathbf{R}''$. Subsequently, Q_1 stores the value $\|\mathbf{R}' - \mathbf{R}''\|_1$ (line 11). If the length of Q_1 reaches $l/2$, Algorithm 1 fits regression lines for all the elements in Q_1, where the independent variable and the dependent variable is the index and the corresponding

element in Q_1 respectively (line 12). Under confidence level $1-\alpha$, we obtain the lower confidence bound κ of the fitted line's slope (line 14). The detailed calculation procedure can be found in [19]. At last, when $\kappa > 0$, Algorithm 2 needs to be re-initialized (lines 15-17). The reason is that Algorithm 2 tends to be stable and fails to respond to the distribution variations when running for a long time [7], [20].

Algorithm 1: Universal Algorithm Framework of Ostasos

Input: The candidate algorithm set $\mathcal{A}$; the size of queue l; the confidence level $1-\alpha$

1 **Initialize** two queues Q_0 and Q_1 with size l and $\frac{l}{2}$, respectively
2 **for** $t = 1; t < T; t++$ **do**
3 Run Algorithm 2, obtain the candidate algorithm A_t and its reward R_{t,A_t}
4 $Q_0.enqueue(A_t, R_{t,A_t})$ %When $Q_0.length() = l$, the enqueue operation leads to dequeue the oldest element.
5 **if** $Q_0.length() = l$ **then**
6 Initialize two empty reward vectors, $\mathbf{R}'$ and $\mathbf{R}''$
7 **foreach** $A \in \mathcal{A}$ **do**
8 Determine the index θ_1 ($\theta_1 < \frac{l}{2}$) and θ_2 ($\theta_2 \geq \frac{l}{2}$) such that $Q_0(\theta_1) = Q_0(\theta_2) = (A, *)$ %If θ_1 or θ_2 is not unique, select one randomly
9 $R'_A = Q_0(\theta_1)(1)$ %R'_A is the reward of A in $\mathbf{R}'$
10 $R''_A = Q_0(\theta_2)(1)$ %R''_A is the reward of A in $\mathbf{R}''$
11 $Q_1.enqueue(\|\mathbf{R}' - \mathbf{R}''\|_1)$
12 **if** $Q_1.length() = \frac{l}{2}$ **then**
13 Linear fit elements in Q_1 with index as independent variables and corresponding values as dependent variables
14 Obtain lower confidence bound κ of the fitted line's slope with confidence level $1-\alpha$ [19]
15 **if** $\kappa > 0$ **then**
16 Restart the Algorithm 2
17 Re-initialize two queues Q_0 and Q_1

Inspired by [20], we propose a UCB-based selector algorithm shown in Algorithm 2. At the beginning of the first execution, we initialize $t' = 0$, $h_i(-1) = 0$, and $q_i(0) = 0$ (line 1) where t' is the epoch index, $h_i(t')$ denotes the total selected times of A^i at epoch t' and $q_i(t')$ is the length of the virtual queue of A^i. Note that these variables can be re-initialized when restarting Algorithm 2. If A^i is selected, we set $\overline{\mu}_i(t')$ as the minimum between the upper confidence bound of reward and D (lines 3-7). Otherwise, the upper confidence bound $\overline{\mu}_i(t')$ is set to be D (line 6). For each A^i, we increase the length of its virtual queue by $r - d_i(t'-1)$ where $d_i(t'-1)$ equals to 1 only when A^i is selected at epoch $t'-1$ (line 7). Next Algorithm 2 selects the candidate algorithm with the maximum sum of $l|\mathcal{A}|\cdot\overline{\mu}_j(t')$ and $D\cdot q_j(t')$ (line 8) and record the selected algorithm by setting $d_j(t') = 1$ (line 9). Once a candidate algorithm is selected, Algorithm 2 can obtain the corresponding reward (line 10). Subsequently, for each A^i, the average reward $\hat{\mu}_i(t')$ and the selected times $h_i(t')$ would be calculated (lines 11-13). At the end, we increase t' by 1 (line 14) and return the selected candidate algorithm and its reward to Algorithm 1 (line 15).

C. Theoretical Analysis

Before analyzing the performance, we first introduce a related property for Algorithm 2.

Algorithm 2: UCB-based Selector Algorithm

Input: The candidate algorithm set $\mathcal{A}$; the queue size l; the minimum selection fraction for each candidate algorithm $r = \frac{|\mathcal{A}|}{0.5l}$; the upper bound of reward D.
Output: The selected candidate algorithm A^j and the reward $R_{t',A_{t'}}$

1 **Initialize** $t' = 0$, $h_i(-1) = 0$ and $q_i(0) = 0$ for each $A^i \in \mathcal{A}$
2 **for** $A^i \in \mathcal{A}$ **do**
3 **if** $h_i(t'-1) > 0$ **then**
4 $\overline{\mu}_i(t') = \min\left\{\hat{\mu}_i(t'-1) + D\cdot\sqrt{\frac{3\log t'}{2h_i(t'-1)}}, D\right\}$ %where $\hat{\mu}_i(t'-1)$ is average reward of A^i in the past $t'-1$ rounds
5 **else**
6 Set $\overline{\mu}_i = D$
7 $q_i(t') = \max\{q_i(t'-1) + r - d_i(t'-1), 0\}$; % $d_i \in \{0,1\}$ is an indicator of whether A^i is played or not in t'
8 Set $A^j = \arg\max_{A^i\in\mathcal{A}} D\cdot q_i(t') + l|\mathcal{A}|\cdot\overline{\mu}_i(t')$
9 Set $d_j(t') = 1$, and for $i \neq j$, set $d_i(t') = 0$
10 Run A^j and obtain the reward $R_{t',A_{t'}}$
11 **for** *each* $A^i \in \mathcal{A}$ **do**
12 $h_i(t') = \sum_{k=0}^{t'} d_i(k)$
13 $\hat{\mu}_i(t') = \frac{\sum_{k=0}^{t'} R_{t',A_{t'}} d_i(k)}{h_i(t')}$
14 $t'++$
15 **Return** A^j and $R_{t',A_{t'}}$.

Property 1. *Algorithm 2 selects each candidate algorithm at least once during $l/2$ epochs,* i.e.*, it can guarantee the existence of θ_1 and θ_2 defined in line 8 of Algorithm 1 [20].*

Based on two properties above, we begin to analyze the competitive ratio of Algorithm 1.

Theorem 1. *The* competitive ratio *of Algorithm 1 is at least* $\max_{A\in\mathcal{A}} CR(A) - O\left(\frac{Regret_S}{T}\right)$*, where $Regret_S$ is the static regret of Algorithm 2. When $T\to\infty$ and the static regret of selector algorithm is sublinear, the* competitive ratio *is at least* $\max_{A\in\mathcal{A}} CR(A)$.

Proof. According to the definition of *competitive ratio*, the *competitive ratio* of Algorithm 1 $CR_1 = \min_{\mathcal{D}} \frac{\sum_{t=1}^{T}\mu_{t,A}}{\sum_{t=1}^{T}\mu_{t,OPT}}$. Combined the definition of *Static Regret*, we have

$$CR_1 = \min_{\mathcal{D}} \frac{\max_{A^*}\sum_{t=1}^{T}\mu_{t,A^*} - Regret_S}{\sum_{t=1}^{T}\mu_{t,OPT}}$$

Since $\sum_{t=1}^{T}\mu_{t,OPT}$ is linear under Assumption 1 and the order of static regret is independent of $\mathcal{D}$, we obtain $\frac{Regret_S}{\sum_{t=1}^{T}\mu_{t,OPT}} = O\left(\frac{Regret_S}{T}\right)$. Accordingly,

$$\begin{aligned} CR_1 &= \min_{\mathcal{D}} \frac{\max_{A^*}\sum_{t=1}^{T}\mu_{t,A^*}}{\sum_{t=1}^{T}\mu_{t,OPT}} - O\left(\frac{Regret_S}{T}\right) \\ &\geq \max_{A^*}\min_{\mathcal{D}} \frac{\sum_{t=1}^{T}\mu_{t,A^*}}{\sum_{t=1}^{T}\mu_{t,OPT}} - O\left(\frac{Regret_S}{T}\right) \\ &= \max_{A^*} CR(A^*) - O\left(\frac{Regret_S}{T}\right) \end{aligned}$$

When the static regret of selector algorithm is sublinear, $\lim_{T\to\infty}\frac{Regret_S}{T} = 0$ so that $\lim_{T\to\infty} CR_1 \geq \max_{A^*} CR(A^*)$. □

Remark: Theorem 1 implies that if the candidate algorithm set $\mathcal{A}$ consists of at least one algorithm with provable *competitive ratio*, then Algorithm 1 has *competitive ratio* guarantee. The *competitive ratio* in Theorem 1 indicates that the performance of Ostasos is affected by both the candidate set and the selector algorithm: the first term $\max_{A\in\mathcal{A}} CR(A)$ denotes the performance of candidate set; the second term $-O\left(\frac{Regret_S}{T}\right)$ denotes the performance of the selector algorithm.

When the distribution keeps unchanged or all changes are detected, the static regret of Algorithm 2 is sublinear [20]. To obtain a sublinear regret in every situation, classic bandit algorithms under the adversary setting, like Exp3, can be simply modified as a selector algorithm. Note that a larger static regret of Exp3 can lead to a higher order $O\left(\frac{Regret_S}{T}\right)$, which is why we modify UCB as a default selector algorithm. Due to the space limitations, we just show the key intuition to modify Exp3, that is, if there are unselected candidate algorithms during $l-1$ epoches, randomly choose one of them with equal probability; otherwise, run Exp3 directly.

We give the following theorem in a ideal case to explain why the restarting policy is designed in this way.

Theorem 2. *Assume that a change of distribution* ($\mathcal{D}_{t_0} \neq \mathcal{D}_{t_0+1}$) *is significant enough so that* $\mathbb{E}\left(\|\mathbf{R}_{x_{t_0}} - \mathbf{R}_{x_{t_0+1}}\|_1\right) \geq \mathbb{E}\left(\|\mathbf{R}_{x'_{t_0}} - \mathbf{R}_{x''_{t_0}}\|_1\right)$*, and the probability of selecting each candidate algorithm* $A \in \mathcal{A}$ *keeps unchanged within* l *epoches. Under Property 1, the length of* Q_0 *reaches* l *after* t_0*, then at epoch* $t_0 + l/2$*, the expectation of each element in* Q_1*,* $\mathbb{E}[Q_1(\lambda)]$*, increases when the index* λ *becomes larger,* i.e.

$$\mathbb{E}[Q_1(\lambda)] = \eta \cdot \lambda + \beta, \tag{2}$$

where $\eta = \frac{\mathbb{E}[Q_1(l/2)]-\beta}{0.5l} > 0$*, and* β *is the expectation of the dequeued element at epoch* $t_0 + l/2$*. When the distribution keeps unchanged,* $\mathbb{E}[Q_1(\lambda)]$ *is a constant.*

Proof. At epoch $t_0 + l/2$, $Q_1(\lambda)$ is an enqueued element at epoch $t_0+\lambda$. At epoch $t_0+\lambda$, the elements in Q_0 ranging from $l-\lambda$ to l stem from $\mathcal{D}_{t_0+1}$ and those ranging from 0 to $l-\lambda-1$ stem from $\mathcal{D}_{t_0}$. When we construct $\mathbf{R}''$ at epoch $t_0 + \lambda$, R''_A stems from either $\mathcal{D}_{t_0}$ or $\mathcal{D}_{t_0+1}$. We introduce $R''_{A^i,0}$ and $R''_{A^i,1}$ to indicate whether R''_A stems from $\mathcal{D}_{t_0}$ or not. Combining the analysis above, we can derive $\Pr[R''_A \text{ stems from } \mathcal{D}_{t_0}] = 1 - \frac{\lambda}{l/2}$ and $\Pr[R''_A \text{ stems from } \mathcal{D}_{t_0+1}] = \frac{\lambda}{l/2}$.

$$\begin{aligned}
&\mathbb{E}[Q_1(\lambda)] = \mathbb{E}\left(\sum_{i=1}^{|\mathcal{A}|} \left|R'_{A^i} - R''_{A^i}\right|\right) \\
=&\frac{l/2-\lambda}{l/2}\mathbb{E}\left(\sum_{i=1}^{|\mathcal{A}|}\left|R'_{A^i} - R''_{A^i,0}\right|\right) + \frac{\lambda}{l/2}\mathbb{E}\left(\sum_{i=1}^{|\mathcal{A}|}\left|R'_{A^i} - R''_{A^i,1}\right|\right) \\
=&\frac{l/2-\lambda}{l/2}\beta + \frac{\lambda}{l/2}\mathbb{E}[Q_1(l/2)]
\end{aligned}$$

Replacing $\eta = \frac{\mathbb{E}[Q_1(l/2)]-\beta}{0.5l}$ and rearranging the above equation, we can derive that Eq. (2) can be established. Since $\mathbb{E}\left(\|\mathbf{R}_{x_{t_0}} - \mathbf{R}_{x_{t_0+1}}\|_1\right) \geq \mathbb{E}\left(\|\mathbf{R}_{x'_{t_0}} - \mathbf{R}_{x''_{t_0}}\|_1\right)$, we conclude that $\mathbb{E}[Q_1(l/2)] - \beta > 0$. Obviously, when the distribution variations keep unchanged, $\mathbb{E}[Q_1(\lambda)]$ is a constant. □

Remark: Theorem 2 implies that the distribution variations can lead to a positive slope, but not vise versa. Hence, we introduce the lower confidence bounds of the slopes to mitigate the above issue [19], *i.e.*, when the lower confidence bound of the slope is greater than *zero*, we assume that the variations happen. Judging from the lower confidence bound of the slope is a heuristic method to reduce wrong judgements.

IV. Applying Ostasos to Online Matching

In this section, we apply Ostasos to online bipartite matching problem, where we use batch and deep Q-learning network approaches to construct our candidate algorithm set.

Online bipartite matching is a classical online optimization that can model many practical problems. Here we generalize online bipartite matching to the case that the duration of each vertex and the weight of each edge can be sampled from a distribution. If a vertex is not matched during its duration, this vertex would leave. Succinctly,

Definition 1. (Generalized Online Bipartite Matching, GOBM). *An online bipartite graph is defined as* $G = (L, R, E)$*, where* $L = \{i|i \in \mathbb{N}\}$ *and* $R = \{j|j \in \mathbb{N}\}$ *are the sets of left and right vertices, and* $E \subset L \times R$ *is the set of edges between* L *and* R*. The duration of each vertex and the weight* e_{ij} *of each edge in* E *is sampled from a distribution* $\mathcal{D}$ *and* $\mathcal{E}$*, respectively. The vertices' arriving frequency is denoted by the probability of sampling a vertex with non-zero duration from* $\mathcal{D}$*. When two vertices are matched, a matching reward (the edges' weight) is produced. The objective is to maximize the total matching reward in* G*.*

In practice, $\mathcal{E}$ usually can be measured, while $\mathcal{D}$ cannot. Taking the online car-hailing for example, the reward is proportional to the measurable trip distance [21] and thus the reward distribution $\mathcal{E}$ is a priori. The accurate waiting time is hard to predict and thus the duration distribution $\mathcal{D}$ is unknown [22]. However, the type of the waiting time can be abstracted from history data [22] in the online car-hailing scenario. The type of the distribution $\mathcal{D}$ and the range of its expectation can be estimated. Now we begin to construct the candidate algorithm set and apply Ostasos to solve the GOBM problem especially in the online car-hailing scenario.

Using Batch algorithm to construct candidate algorithm set. As mentioned in Sec. II, the Batch algorithm transforms the online bipartite matching problem into a batch partition, where the batch size is the length of a pre-defined time interval. Batch size affects the matching quality greatly and the optimal batch size varies with the different duration distributions.

Theorem 3. *For GOBM problem with the distribution* $\mathcal{D}$ *and* $\mathcal{E}$*, if the batch size of Batch algorithm is* b*, the expectation of the average reward* $\overline{\mathcal{T}_b}$ *is:*

$$\mathbb{E}\left(\overline{\mathcal{T}_b}\right) \geq \frac{1}{b}\sum_{m=1}^{b}\sum_{n=1}^{b}\xi_m^b\xi_n^b \sum_{j=1}^{\min\{m,n\}} \mathbb{E}\left(\max_{e\in E'_j} e\right)$$

where E'_j represents a edge set and ξ_v^u is a probability, both of which are defined in Appendix A.

The detailed proof can be found in Appendix A.

Remark: For each possible $\mathcal{D}$, according to Theorem 3, we obtain the optimal batch size b^*, *i.e.*,

$$b^* = \arg\max_b \frac{1}{b} \sum_{m=1}^{b} \sum_{n=1}^{b} \xi_m^b \xi_n^b \sum_{j=1}^{\min\{m,n\}} \mathbb{E}\left(\max_{e \in E'_j} e\right)$$

Based on Theorem 3, the *competitive ratios* of different batch sizes can be further analyzed. For convenience, we use CR_b to denote the *competitive ratio* of the Batch algorithm with batch size b.

Theorem 4. *When the batch size is b, the competitive ratio of the Batch algorithm has a lower bound:*

$$CR_b \geq \min_{\mathcal{D}} \left(\frac{1}{b\mathbb{E}(\mathcal{T}^*)} \sum_{m=1}^{b} \sum_{n=1}^{b} \xi_m^b \xi_n^b \sum_{j=1}^{\min\{m,n\}} \mathbb{E}\left(\max_{e \in E'_j} e\right) \right)$$

where $\mathbb{E}(\mathcal{T}^*)$ is defined in Lemma 2 in Appendix B, $d_{\max}$ is the upper bound of the duration, $p(d)$ denotes the probability of sampling duration d from $\mathcal{D}$, $C_d^k = \binom{d}{k}$.

The detailed proof can be found in Appendix B.

Using trained-DQNs to construct candidate algorithm set. Under a specific duration distribution, the matching of GOBM problem can be formulated as a Markov decision process [6]. One alternative is to use the deep Q-learning networks [5], which updates the Q-value as follows.

$$Q(s_\gamma, a_\gamma) \leftarrow Q(s_\gamma, a_\gamma) + \tau[r_{\gamma+1} + \psi \max_a Q(s_{\gamma+1}, a) - Q(s_\gamma, a_\gamma)]$$

where subscript γ indicates different time steps, s_γ is the state in γ, a_γ is the chosen action, $r_{\gamma+1}$ is the obtained reward after taking action a_γ, τ and ψ are two hyper-parameters. To show how to train a deep Q-learning network for GOBM problem, we introduce the details. (i) **Action Space:** the action $a_\gamma \in \{0, 1\}$ indicates whether we match all survival vertices or not in time step γ. (ii) **Reward:** If $a_\gamma = 1$, $r_{\gamma+1}$ is the produced reward of matching all survival vertices. Otherwise, $r_\gamma = 0$. (iii) **State Representation:** We set the state $s_\gamma = \{n_\gamma^l, n_\gamma^r, M'_\gamma, \overline{T}_\gamma^l, \overline{T}_\gamma^r\}$ where n_γ^l (n_γ^r) represent the number of survival vertices in the left (right) side, M'_γ indicates the obtained reward if we choose a matching action, and $\overline{T}_\gamma^l$ ($\overline{T}_\gamma^r$) denote the average survival time of vertices in the left (right) side.

Based on the input state $s_\gamma = \{n_\gamma^l, n_\gamma^r, M'_\gamma, \overline{T}_\gamma^l, \overline{T}_\gamma^r\}$, DQNs decide to match vertices in the bipartite graph or not. After that, $(s_\gamma, a_\gamma, r_\gamma, s_{\gamma+1})$ is stored and used for the training. The whole training procedure tries to learn from the optimal static batch size on the fly. Hence, we can use the optimal batch size at the beginning to further improve the performance.

Applying Ostasos to dynamically choose the candidate algorithm. Taking Batch algorithms and trained-DQNs as the candidate algorithms, we can select the candidate algorithm by running Algorithm 2 and obtain a reward. According to the records of Q_1 in Algorithm 1, we can detect the distribution variations and restart Algorithm 2 to improve the performance.

Obviously, according to Theorem 1 and Theorem 4, we can analyze *competitive ratio* of Ostasos for GOBM problem.

Corollary 1. *The competitive ratio of Algorithm 1 for GOBM problem is at least $\max_b CR_b - O\left(\frac{Regret_S}{T}\right)$ where CR_b is given in Theorem 4.*

V. EVALUATION

We evaluate Ostasos with trace-driven experiments in the online car-hailing scenario.

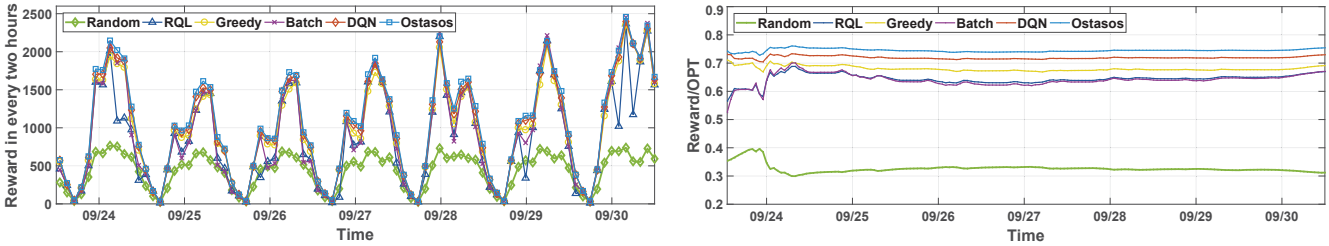
Setup: The dataset we used comes from Didi Chuxing [23] published by its GAIA initiative [24]. The data includes the complete information of online car-hailing problem, *i.e.*, the passengers' requests, the drivers' responses, the waiting time of passengers, and the cost of the orders. The candidate algorithm set $\mathcal{A}$ in Ostasos is constructed by Batch algorithms with batch sizes calculated by Theorem 3 and trained-DQNs.

Comparing Ostasos with Single Algorithms. We compare Ostasos with following benchmark single algorithms:

- **Greedy:** Greedy only cares about the current reward;
- **RQL:** RQL uses a restricted Q-learning approach to learn how to choose the best matching over the time horizon;
- **Batch:** The batch size of the Batch algorithm here is the monthly average waiting time in September;
- **DQN:** DQN used here is the DQN mentioned in Sec. IV without any previous training and learns from the environment continuously;
- **Random:** Random matches passengers and drivers randomly.

The experiment result is shown in Fig. 2(a), where the tick label in the horizontal axis, like "09/25", denotes the time point 12:00 p.m. in that day. For clarity, we only show the total rewards in every two hours. We observe that Ostasos achieves the highest reward in nearly every period. Since Random matches the graph without considering reward, Random gets the lowest rewards. However, we observe that RQL sometimes gets even lower reward than Random, and in Sec. II we have explained the phenomenon: when distribution sharply changes, the more RQL learned from the distribution before, the worse result RQL gets. This issue also occurs in DQN, which causes DQN to never get the highest reward in our experiment. We also observe that the Batch algorithm achieves highest rewards in some periods, while in some other periods (*e.g.* the noons in 09/25 and 09/26), it gets much lower rewards, which indicates that a static batch size only suits some distributions, and this result also aligns with Theorem 3. To reflect how *competitive ratios* vary with time, we show the ratios of accumulated rewards and offline optimum in Fig. 2(b). As time increases, the ratio of Ostasos first fluctuates and then tends to be stable, which conforms to Theorem 1. Also, the ratio of Ostasos is larger than any single algorithms' ratios.

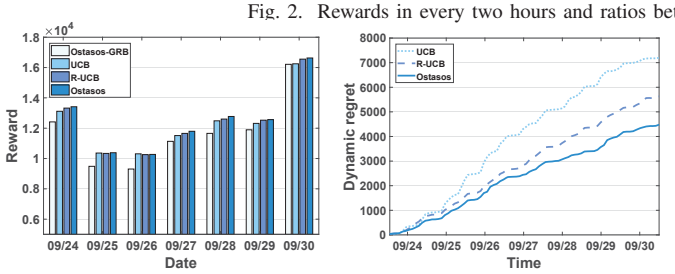
Comparing Ostasos with Other Algorithm-selecting Schemes. We compare Ostasos with the following benchmark schemes:



(a) Rewards of single algorithms and Ostasos in every two hours

(b) Ratios between accumulated rewards and offline optimum

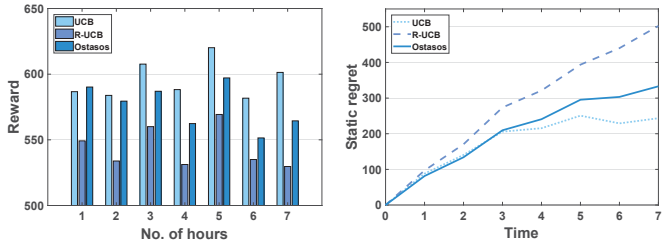
Fig. 2. Rewards in every two hours and ratios between accumulated reward and offline optimum



(a) Rewards in every day

(b) Dynamic regret

Fig. 3. Reward and dynamic regret comparisons.



(a) Rewards in every hour

(b) Static regret

Fig. 4. Reward and static regret comparisons.

- **Ostasos-GRB**: The algorithm set $\mathcal{A}$ in Ostasos-GRB is constructed by Greedy, RQL, and Batch algorithm of which the batch size is the monthly average waiting time in September, other than that, it is identical to Ostasos;
- **UCB** [17]: A widely used multi-arms bandit algorithm where the arms set corresponds to the candidate algorithm set in Ostasos;
- **R-UCB** [7]: R-UCB restarts UCB periodically, other than that, R-UCB is identical to UCB.

Since in a short time the differences among algorithm-selecting schemes are small, to highlight the differences, in Fig. 3(a), we show the rewards of each scheme for each day. In Fig. 3(a), Ostasos-GRB gets the lowest reward in every day. It indicates that compared with using the pre-existing algorithms to construct $\mathcal{A}$, using Batch algorithms with different batch sizes and DQN to construct $\mathcal{A}$ as proposed in this paper is more effective for the GOBM problem. In Fig. 3(a), given that UCB always gets lower rewards than R-UCB, restarting policy indeed improves the performance of classical UCB algorithm when distribution varies with time. Moreover, since Ostasos achieves higher reward than R-UCB in Fig. 3(a), it demonstrates that the restarting policy of Ostasos results in a better performance than the periodically restarting policy does.

To measure the performance of algorithm-selecting schemes when the distribution varies, we introduce *dynamic regrets*. *Dynamic Regret* [10] is the difference between the reward of actual choices and that of the optimal choices at each epoch,

$$Regret_D = \max_{A_t^*} \left(\sum_t^T R_{t,A_t^*} - \sum_t^T R_{t,A_t} \right)$$

where A_t is the selected candidate algorithm at each epoch t, and A_t^* is the optimal algorithm at each epoch t.

We show the *Dynamic Regrets* of algorithm-selecting schemes with same $\mathcal{A}$ (*i.e.* UCB, R-UCB, and Ostasos) in Fig. 3(b). In Fig. 3(b), the different increasing rates of *Dynamic Regrets* indicate that if the algorithm-selecting scheme has a restarting policy when distribution varies with time, we get a lower regret, *i.e.*, the regrets of Ostasos and R-UCB are much lower than UCB. Obviously, Ostasos achieves a much lower *Dynamic Regret* than UCB and R-UCB.

When the distribution is static, comparing to UCB, algorithm-selecting schemes with restarting policy cause extra loss. Hence, it is necessary to evaluate the algorithm-selecting schemes when the distribution is static. In Fig. 4(a), we evaluate UCB, R-UCB, and Ostasos using a dataset where the duration distribution keeps unchanged and is based on the statistics of the real dataset at 10:00 a.m. to 5:00 p.m (working time). In Fig. 4(a), since UCB never restarts itself and learns from all history results, the hourly reward gaps between UCB and other schemes increase; on the contrary, since R-UCB restarts UCB periodically, it repeats to explore arms after restarting so that the hourly reward of R-UCB is lowest. For Ostasos, the restarting policy in Algorithm 1 restarts Algorithm 2 much less often when the distribution is static so that the reward of Ostasos in every hour is close to the reward of UCB.

Since the distribution is static, we use *Static Regret* to measure the performance of algorithm-selecting schemes. We show the corresponding *Static Regrets* of algorithm-selecting schemes in Fig. 4(b). In the beginning, the increasing rates of UCB and Ostasos are close, and R-UCB has the fastest increasing rate, which verifies that the restarting policy causes extra loss when the distribution keeps unchanged. Then, since UCB never restarts itself, the increasing rate of UCB's *Static Regret* becomes lower than Ostasos's. Note that the regret gap between UCB and Ostasos is much lower than the gap between UCB and R-UCB, Ostasos has much better performance than R-UCB when the distribution keeps unchanged.

TABLE I
True positive rates and false positive rates under different confidence levels.

Confidence level $(1-\alpha)$	95.00%	99.00%	99.50%	99.90%
True positive rate	92.59%	92.31%	88.89%	88.65%
False positive rate	5.14%	3.98%	3.73%	1.82%

The Performance of The Restarting Policy in Ostasos. We use true positive and false positive rates to evaluate the restarting policy. Here, the true positive rate is the ratio between the number of correct restarts and the number of distribution variations; the false positive rate is the ratio between the number of wrong restarts and the number of detections when the distribution is static. Hence, a good restarting policy requires a high true positive rate and a low false positive rate.

Since the confidence level $(1-\alpha)$ has a significant impact on the performance of the restarting policy, in Table I, we show the true positive and false positive rates when we run Ostasos in the real dataset with different confidence levels. In Table I, the high true positive rates and low false positive rates validate the effectiveness of the restarting policy. We also observe that when confidence level increases, the true positive and false positive rates decrease, which can be explained briefly as follows. For a distribution variation, under a higher confidence level, the lower confidence bound κ of the slope in Ostasos is smaller than the lower bound under a lower confidence level so that a slight variation may not be detected under a high confidence level; similarly, when the distribution is static, under a higher confidence level, the smaller lower confidence bounds reduce the wrong restarts. Hence, by tuning the confidence level, we can control the detection range of variations of the restarting policy, *i.e.*, if we only need to recognize considerable variations, a high confidence level should be set.

VI. Related Work

We briefly review the prior art on non-stationary online optimization and online bipartite matching.

Non-stationary online optimization: In the non-stationary online learning problem, [8] considers offline optimal dynamic solution and introduces *dynamic regret* in stochastic approximation. It has been proved that *dynamic regret* is sublinear only if the distribution variations — *variation budget* or the times of changes — is bounded [7], [10], [12], [25], [26]. To solve non-stationary online learning problem, one alternative is to restart the classical online learning algorithms periodically, like *online gradient descent* [8], [25] and *bandit algorithm (UCB)* [7]. Instead of restarting algorithm according to the change of distribution, the restarting policy in above works just restarts algorithms periodically, which leads to a lot of unnecessary restarts causing extra loss. Also the idea of *ensemble learning* can take advantage of different algorithms in various environments [1], [18], [27]. However it is not suitable for the problems in which the setting is bandit and the candidate algorithms need feedback to update their parameters. [28] and [11] design an automatic selecting framework respectively, to select the most suitable one when network conditions change. Their approaches are specific to congestion control and resource management.

Online Bipartite Matching: Online bipartite matching problem can be roughly divided into one-sided and two-sided online settings. Ranking algorithm [29] is a classical 0.6534-competitive one-sided online algorithm. Furthermore, the work in [30] considers two-sided online case and the competitive ratio is 0.25. This bound is further improved to be 0.47 [4]. Later in [6], a $\frac{1}{C}$-competitive two-sided online algorithm is proposed, where C is an upper bound of vertices' duration. The work in [13] generalizes the online bipartite matching to the arbitrary graph matching.

VII. Conclusion

In this paper, we study online optimization problems with dynamic distributions of variables. To solve this problem, we propose an automatic algorithm selection framework, Ostasos, with a provable *competitive ratio* guarantee. Combining Batch algorithms and trained-DQNs to construct the algorithm set, we apply Ostasos in two-sided online bipartite matching problem in which the duration distribution varies with time. The evaluation results based on a real dataset demonstrate that Ostasos can solve two-sided online bipartite matching problem very well. In the future, we plan to apply Ostasos to solve non-stationary online optimization problems in other fields.

Acknowledgment

We thank the anonymous reviewers for their helpful comments on drafts of this paper. This work was supported in part by the China NSF grants (61802172), China NSF of Jiangsu Province (BK20201248) and Open Fund of PDL (WDZC20205500109).

Appendix

A. Proof of Theorem 3

As the produced reward depends on the bipartite graph at the matching moment. We use $\mathcal{T}(m,n)$ to denote the total reward using Batch algorithm where $m=|L|$ and $n=|R|$.

Lemma 1. *For the Batch algorithm, there exists a set sequence, E'_1, E'_2, $\cdots$, $E'_{\min\{m,n\}}$, and satisfies*

$$\mathbb{E}[\mathcal{T}(m,n)] \geq \sum_{j=1}^{\min\{m,n\}} \mathbb{E}\left(\max_{e\in E'_j} e\right)$$

where edge's weight $e \sim \mathcal{E}$.

Proof. For convenience, the set of vertices in the left side is denoted by $L=\{1,2,...,m\}$ and that in the right side is denoted by $R=\{1,2,...,n\}$. Without loss of generality, we assume $m \leq n$. This means that all m vertices in the left side are matched. We describe the following greedy matching process to construct E'_1, E'_2, $\cdots$, $E'_{\min\{m,n\}}$. We first choose the edge with the largest weight for the first vertex in the left side. There are n vertices in the right side and the set $E'_1=\{e_{11},e_{12},...,e_{1n}\}$. Accordingly, the expectation of the reward is $\mathbb{E}\left(\max_{e\in E'_1} e\right)$. Assume the matched vertex in the

right side is i. At this point, there are only $n-1$ vertices in the left side and $E'_2 = \{e_{21}, ..., e_{2n}\} - \{e_{2i}\}$. Likewise, we can construct $E'_3, \cdots, E'_{\min\{m,n\}}$ one by one. Hence, the expectation of total reward is $\sum_{j=1}^{m} \mathbb{E}\left(\max_{e\in E'_j} e\right)$. □

Certain vertices may be unmatched in one batch and we use the remain model and the discard model to distinguish whether the unmatched vertices are included in the next batch [6]. Accordingly, we denote the total reward under the remain model by $\mathcal{T}$, and that under discard model by $\mathcal{T}_{discard}$. Since $\mathcal{T}_{discard} \leq \mathcal{T}$ [6], we can analyze the reward under discard model to bound that in the remain model.

Here we slightly abuse the notation and use q to represent both the arrival time of a vertex and this vertex itself in GOBM problem. To estimate the reward in the discard model under a specific batch size by Lemma 1, we first estimate the number of vertices at the matching moment. When the deadline of vertex q is ahead of this matching moment, we call q is out-of-date. We set $F_d(\cdot) = \sum_{j=0}^{d} p(j)$ as probability distribution function of $\mathcal{D}$ where $p(j)$ is the probability that a vertex's duration is j. The last matching moment is denoted by σ. Hence, for the vertex q, the out-of-date probability is $F_d(\sigma + b - q)$. And the probability that a vertex arriving at moment q still survives at the matching moment is $1 - F_d(\sigma + b - q)$, which means the probability distribution is Bernoulli when q is specified. As $1 - F_d(\sigma + b - q)$ changes with q, the Bernoulli distributions varies at different moments. When the batch size b is determined, the probability that when u vertices arrive and v vertices survive at the matching moment is equal to the probability that the sum of u independent and non-identical random indicators is v, where each indicator is a Bernoulli random variable with individual success probability. When we set the sum of independent and non-identical random indicators as X and the number of total indicators as Y,

$$\Pr(v \text{ vertices survive}|u \text{ vertices arrive}) = \Pr(X = v|Y = u)$$

Here, we introduce Poisson binomial distribution.

Definition 2. *[31] The Poisson binomial distribution is the distribution of the sum of independent and non-identical random indicators. Each indicator follows a Bernoulli distribution with individual success probability.*

Obviously, the numbers of survival vertices at the matching moment, m and n, obey Poisson binomial distribution. The probability mass function of Poisson binomial distribution is first formed by the enumeration method [32]. [33] and [31] form the closed-form expression of probability mass function of Poisson binomial distribution. When the number of variables following different Bernoulli distributions is u and the sum of u independent and non-identical random indicators is v, the probability mass function is

$$\xi_v^u = \frac{1}{u+1} \sum_{s=0}^{u} \exp(-\boldsymbol{i}\omega s v) \prod_{j=1}^{u} [1 - p_j + p_j \exp(\boldsymbol{i}\omega s)]$$

where ξ_v^u is the probability that there are v success variables, $\omega = 2\pi/(u+1)$, $\boldsymbol{i} = \sqrt{-1}$, and p_j is the success probability. When batch size is b, $p_j = F_d(\sigma + b - j)$. We can obtain

$$\Pr(v \text{ vertices survive}|u \text{ vertices arrive}) = \xi_v^u$$

Therefore, the expectation of the total reward when the matching in one batch with size b finishes is

$$\mathbb{E}(\mathcal{T}_{discard}) = \sum_{m=1}^{b} \sum_{n=1}^{b} \mathcal{T}(m,n) \xi_m^b \xi_n^b \tag{3}$$

Combining Lemma 1, Eq. (3) and $\mathcal{T}_{discard} \leq \mathcal{T}$ [6], we obtain the following inequation which concludes the proof.

$$\mathbb{E}(\overline{\mathcal{T}_b}) \geq \frac{1}{b} \sum_{m=1}^{b} \sum_{n=1}^{b} \xi_m^b \xi_n^b \sum_{j=1}^{\min\{m,n\}} \mathbb{E}\left(\max_{e\in E'_j} e\right)$$

B. Proof of Theorem 4

To give a *competitive ratio* analysis, we need to get an upper bound of the offline optimal.

Lemma 2. *Arbitrarily choose a vertex q with duration d in the left side, whose arriving time is also denoted as q for convenience. The set of the vertices in the right side surviving between the interval $[q, q+d]$ are denoted as $\mathcal{R}_q$. The maximum reward of matching q is $\mathcal{T}^* = \max_{j\in\mathcal{R}_q} e_{qj}$. The expectation of $\mathcal{T}^*$ is*

$$\mathbb{E}(\mathcal{T}^*) = \sum_{d=0}^{d_{\max}} \sum_{i=0}^{d_{\max}} p(d) \xi_i^{d_{\max}} \sum_{k=0}^{d_{\max}} \mathbb{E}\left(\max_{e\in E'_{i+k}} e\right) \times C_d^k [p(0)]^{d-k} [1-p(0)]^k$$

Proof. The vertices in $\mathcal{R}_q$ can be divided into two parts. (i) the vertices arrive before q and still survive when q arrives. The number of these vertices is denoted by i. The probability that there are i survival vertices in right side at q is $\xi_i^{d_{\max}}$, where $d_{\max}$ is an upper bound of vertex's duration. (ii) the vertices arrive after q and still survive at $q+d$. The number of these vertices is denoted by k. k depends on not only the arriving frequency $1 - p(0)$, but also q's duration d. Once d is determined, the probability distribution of k is a binomial distribution, *i.e.*, the corresponding probability is $C_d^k [p(0)]^{d-k} [1-p(0)]^k$. Since the number of vertices in right side is $k+i$, the expectation of the matching reward is $\mathbb{E}(\max_{e\in E'_{i+k}} e)$. Based on the analysis above, we conclude the proof of Lemma 2. □

Since $\mathbb{E}(\mathcal{T}^*)$ is an upper bound of the expectation of the reward obtained by matching a pair of vertices, the expectation of average reward obtained by the offline optimum is

$$\mathbb{E}\left(\overline{OPT}\right) \leq \mathbb{E}(\mathcal{T}^*)$$

where $\overline{OPT}$ denotes the offline optimum.

When the batch size is b, the *competitive ratio* of Batch algorithm is expressed as

$$CR_b = \frac{\mathbb{E}(\mathcal{T}/b)}{\mathbb{E}\left(\overline{OPT}\right)} \geq \frac{\mathbb{E}(\overline{\mathcal{T}_b})}{\mathbb{E}(\mathcal{T}^*)} \tag{4}$$

Theorem 3, Lemma 2, and Eq. (4) conclude the proof.

REFERENCES

[1] K. Pang, M. Dong, Y. Wu, and T. M. Hospedales, "Dynamic ensemble active learning: A non-stationary bandit with expert advice," in *IEEE ICPR*, 2018, pp. 2269–2276.

[2] Hazan and Elad, "Introduction to online convex optimization," *Foundations and Trends in Machine Learning*, vol. 2, no. 3-4, pp. 157–325, 2016.

[3] Z. Y. W. D. Zhou, P. and H. Jin, "Differentially private online learning for cloud-based video recommendation with multimedia big data in social networks," *IEEE transactions on multimedia*, vol. 18, no. 2, pp. 1217–1229, 2016.

[4] J. P. Dickerson, K. A. Sankararaman, A. Srinivasan, and P. Xu, "Assigning tasks to workers based on historical data: Online task assignment with two-sided arrivals," in *Springer AAMAS*, 2018, pp. 318–326.

[5] 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*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *IEEE ICDE*, 2019, pp. 1478–1489.

[7] C.-Y. Wei, Y.-T. Hong, and C.-J. Lu, "Tracking the best expert in non-stationary stochastic environments," *Advances in neural information processing systems*, vol. 29, pp. 3972–3980, 2016.

[8] O. Besbes, Y. Gur, and A. Zeevi, "Non-stationary stochastic optimization," *Operations research*, vol. 63, no. 5, pp. 1227–1244, 2015.

[9] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *Springer ALT*, 2011, pp. 174–188.

[10] L. Zhang, T. Yang, Z.-H. Zhou *et al.*, "Dynamic regret of strongly adaptive methods," in *ACM ICML*, 2018, pp. 5882–5891.

[11] J. Comden, S. Yao, N. Chen, H. Xing, and Z. Liu, "Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms," in *ACM POMACS*, 2019, pp. 1–30.

[12] X. Chen, Y. Wang, and Y.-X. Wang, "Nonstationary stochastic optimization under l p, q-variation measures," *Operations Research*, vol. 67, no. 6, pp. 1752–1765, 2019.

[13] I. Ashlagi, M. Burq, C. Dutta, P. Jaillet, A. Saberi, and C. Sholley, "Edge weighted online windowed matching," in *ACM EC*, 2019, pp. 729–742.

[14] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[15] A. Mehta *et al.*, "Online matching and ad allocation," *Foundations and Trends® in Theoretical Computer Science*, vol. 8, no. 4, pp. 265–368, 2013.

[16] L. L. H. Andrew, S. Barman, K. Ligett, M. Lin, A. Meyerson, A. Roytman, and A. Wierman, "A tale of two metrics: Simultaneous bounds on competitiveness and regret," in *Springer COLT*, 2013, pp. 741–763.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[18] L. Zhang, S. Lu, and Z.-H. Zhou, "Adaptive online learning in dynamic environments," *Advances in neural information processing systems*, pp. 1323–1333, 2018.

[19] DeGroot and MorrisH, *Probability and Statistics*. Addison-Wesley Publishing Company, 1986.

[20] F. Li, J. Liu, and B. Ji, "Combinatorial sleeping bandits with fairness constraints," in *IEEE INFOCOM*, 2019, pp. 1702–1710.

[21] Z. B. L. Fei S, "Research on trip distance distribution model and parameter," *Journal of Traffic and Transportation Engineering*, vol. 2, 2008.

[22] W. T. Wei H, Wang Y *et al.*, "Zest: a hybrid model on predicting passenger demand for chauffeured car," in *ACM CIKM*, 2016, pp. 2203–2208.

[23] DiDi chuxing. URL: https://www.didiglobal.com/.

[24] Gaia. URL: https://outreach.didichuxing.com/research/opendata/.

[25] A. Mokhtari, S. Shahrampour, A. Jadbabaie, and A. Ribeiro, "Online optimization in dynamic environments: Improved regret rates for strongly convex problems," in *IEEE CDC*, 2016, pp. 7195–7201.

[26] N. B. Keskin and A. Zeevi, "Chasing demand: Learning and earning in a changing environment," *Mathematics of Operations Research*, vol. 42, no. 2, pp. 277–307, 2017.

[27] Y. Shen, T. Chen, and G. B. Giannakis, "Online ensemble multi-kernel learning adaptive to non-stationary and adversarial environments," in *JMLR AISTATS*, 2018.

[28] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei, "Dynamic tcp initial windows and congestion control schemes through reinforcement learning," *IEEE Journal on Selected Areas in Communications*, pp. 1231–1247, 2019.

[29] Z. Huang, Z. G. Tang, X. Wu, and Y. Zhang, "Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals," *arXiv preprint arXiv:1804.07458*, 2018.

[30] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *IEEE ICDE*, 2016, pp. 49–60.

[31] Y. Hong, "On computing the distribution function for the sum of independent and non-identical random indicators," Tech. Rep., 2011.

[32] Y. H. Wang, "On the number of successes in independent trials," *Statistica Sinica*, vol. 3, pp. 295–312, 1993.

[33] M. Fernández and S. Williams, "Closed-form expression for the poisson-binomial probability density function," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 46, no. 2, pp. 803–817, 2010.