Online Matching with Stochastic Rewards: Provable Better Bound via Adversarial Reinforcement Learning

Qiankun Zhang [*1] **Aocheng Shen** [*1] **Boyu Zhang** [1] **Hanrui Jiang** [1] **Bingqian Du** [2]

Abstract

For a specific online optimization problem, for example, online bipartite matching (OBM), research efforts could be made in two directions before it is finally closed, i.e., the optimal competitive online algorithm is found. One is to continuously design algorithms with better performance. To this end, reinforcement learning (RL) has demonstrated great success in literature. However, little is known on the other direction: whether RL helps explore how hard an online problem is. In this paper, we study a generalized model of OBM, named online matching with stochastic rewards (OMSR, FOCS 2012), for which the optimal competitive ratio is still unknown. We adopt an adversarial RL approach that trains two RL agents adversarially and iteratively: the algorithm agent learns for algorithms with larger competitive ratios, while the adversarial agent learns to produce a family of hard instances. Through such a framework, agents converge at the end with a robust algorithm, which empirically outperforms the state of the art (STOC 2020). Much more significantly, it allows to track how the hard instances are generated. We succeed in distilling two structural properties from the learned graph patterns, which remarkably reduce the action space, and further enable theoretical improvement on the best-known hardness result of OMSR, from 0.621 (FOCS 2012) to 0.597. To the best of our knowledge, this gives the first evidence that RL can help enhance the theoretical understanding of an online problem.

[*]Equal contribution [1]School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China [2]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. Correspondence to: Qiankun Zhang <qiankun@hust.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Optimization in an online setting is one of the most significant research branches in combinatorial optimization, because it captures more practical real-world applications by considering sequential variables or constraints arriving over time. For example, in online advertising, a search engine like Google must select an advertiser to show for a search request, without any knowledge of the future requests that may follow. This scenario is intrinsically modeled by the *online bipartite matching* (OBM) problem proposed by Karp et al. (1990). To deal with the challenge of the uncertainty of inputs, theoretical studies (See Mehta (2013) for a survey) constantly improve the ratio between the (expected) solution given by an algorithm and the optimal solution to a corresponding offline benchmark, which is called *competitive ratio*. Recently, there has been a line of research using machine learning (ML) techniques to solve online optimization problems, among which, *reinforcement learning* (RL) achieves remarkable success in finding high-performance algorithms (Du et al., 2022a; Kong et al., 2019; Wang et al., 2019; Alomrani et al., 2021). These successes essentially stem from the similarities in sight between RL and online algorithms. Both of them deal with sequences of inputs, and target on finding strategies that optimize (maximize or minimize) their objectives (or cumulative rewards).

However, almost all existing works target training algorithms for online optimization problems, for which the optimal competitive ratios have been known. In other words, their results indicate that the RL agents can eventually converge to perform the best on the worst-case inputs, and the learned policies are broadly consistent with the best-known theoretical algorithms (e.g., the online primal-dual algorithms). These attempts are significant but preliminary. There is a large class of open-ended online optimization problems, which may face critical challenges from two directions: one is to *find robust algorithms* as previous works, corresponding to improve the *lower bound* of competitive ratio; another one is to *construct hard instances*, corresponding to an *upper bound*, such that no algorithm can achieve a competitive ratio better than that. Applying RL approaches to such problems raises the following interesting questions:

Our Research Questions

- Can RL help enhance our (theoretical) understanding in the hardness of an online problem?
- Can and how can RL be applied to an unclosed online problem?

This paper gives an attempt to address these issues. We focus on a concrete problem, named *Online Matching with Stochastic Rewards* (OMSR) proposed by Mehta & Panigrahi (2012), which generalizes the classic OBM model. This model is more practical to the online advertising platform, where payments are given by advertisers only when their ads are clicked by a user. The click-through-rate can be used as an estimation of the probability of those clicks. Only when the user actually clicks the ad, which is dominated by a stochastic process, the advertiser should pay for it. The randomness coming from the problem itself brings remarkable difficulties in theoretical analysis. While an optimal $1-1/e \approx 0.632$-competitive algorithm is known to OBM, OMSR remains open: an upper bound of 0.621 [1] (Mehta & Panigrahi, 2012) and a lower bound of 0.572 [2] (Huang & Zhang, 2020) are known the best.

Our paper studies OMSR and considers both directions in learning robust algorithms and hard instances. We set up an *adversarial reinforcement learning* (Pinto et al., 2017) framework for OMSR. The framework consists of an iterative process between *an adversary agent* (`adv`) and *an algorithm agent* (`alg`): `adv` learns the hardest instances which make the current `alg` perform the worst; `alg` then learns for a better performance over these hard instances [3]. Besides obtaining a robust algorithm for OMSR as what previous works have done, much more significantly and interestingly, we observe two structural properties from graph patterns of hard instances learned by `adv`, named a *consistency* property and an *exclusivity* property. These properties not only help reduce the action space in training `adv`, but further serve as ingredients for theoretically proving an improved upper bound for OMSR. Precisely, our main results are concluded as follows:

- We prove that there is no algorithm for OMSR with a competitive ratio of more than 0.597, beating the best known upper bound of 0.621 (Mehta & Panigrahi, 2012).
- We empirically show that our algorithms learned by `alg` performs better than the state-of-the-art (SOTA).

Extensive experiments are conducted to evaluate both hard instances and algorithms learned through our framework.

1.1. Related Works

Theoretical results for OMSR. Like the well-known online optimization problem, AdWords (Mehta et al., 2005; Huang et al., 2020), OMSR is also a variant of the online bipartite matching problem, which is first proposed by Mehta & Panigrahi (2012). They give an upper bound of 0.621 for OMSR as a hardness result. Also, they propose two algorithms, named Balance and Ranking, for a special case when success probabilities are all equal, and prove that they are 0.567 and 0.534-competitive respectively. For the unequal probability case, Mehta et al. (2014) prove that Balance can achieve 0.534-competitive. Later, these bounds are improved by Huang & Zhang (2020). They prove that the competitive ratios of the Balance algorithm are 0.576 for equal success probabilities and 0.572 for unequal case. Their results are restricted to small success probabilities (e.g., smaller than 0.01), which is close to the real click-through-rate in an internet advertising platform. Meanwhile, another series of work (Goyal & Udwani, 2019; Huang et al., 2023) also analyzes Balance and Ranking but against a weaker benchmark. In this paper, we evaluate our algorithms in both equal and unequal success probability settings, varying from 0 to 1, by viewing Balance as a baseline.

Reinforcement learning for online optimization. These series of work are the most relevant to ours, and can be roughly divided into two main categories. One is about designing high-performance algorithms towards *real-world datasets*. Wang et al. (2019) propose a Q-learning for node batches, for a dynamic generalization of OBM; Alomrani et al. (2021) train an algorithm agent based on historical data, and evaluate it on two generalizations of OBM: edge-weighted OBM and online submodular maximization. The other line is about designing *robust* algorithms using RL towards worst-case inputs. Kong et al. (2019) find optimal algorithms for three well-studied online problems, relying on prior knowledge of hard instances. They claim that the RL agent can learn the behavior as theoretically optimal algorithms, i.e., the primal-dual algorithms. Zuzic et al. (2020) propose a GAN-like framework based on Yao's Lemma. They work on AdWords problem but whether their framework can be used for those unclosed problems is not yet clear. Compared to these works, our paper is new in: (1) studying an open-ended online problem; (2) utilizing an adversarial RL approach to model online optimizations as a game; (3) obtaining theoretical bounds with an assistance of learned patterns; (4) generalization ability to other online problems. Besides, using RL to solve (offline) combinato-

[1] Meaning that OBSR is strictly harder than OBM.

[2] This bound is for a restricted case of stochastic rewards problem where the success probabilities over edges are infinitesimal. See Section 2 for details.

[3] More precisely, for robustness, `alg` should be trained over a mixture of hard instances and some randomly generated instances. See Section 4 for details.

rial optimization problems is not young (Bello et al., 2016; Zhang et al., 2020a; Nazari et al., 2018; Kool et al., 2018; Khalil et al., 2017; Du et al., 2022b; Boutilier & Lu, 2016; Shao et al., 2022). We refer to surveys (Mazyavkina et al., 2020; Bengio et al., 2018; Yang & Whinston, 2020) for readers interested in this area.

Adversarial reinforcement learning. Our idea of adversarial RL is similar to Pinto et al. (2017), who train an agent to operate in the presence of a destabilizing adversary. They jointly train the adversary to learn an optimal destabilization policy. Subsequently, follow-up work targets on finding its applications, for example, in video captioning (Hua et al., 2022), grasping moving objects (Wu et al., 2022), robotics (Jiang et al., 2021), and so forth (Ma et al., 2018; Zhang et al., 2020b; Spooner & Savani, 2020; Gisslén et al., 2021). The idea behind online algorithm design is exactly adversarially constructing hard instances while improving algorithms. That is why our methods are intuitively effective. Moreover, for robustness, Vinitsky et al. (2020); Dong et al. (2023) train agents against multiple adversaries instead of a single one. We utilize a similar but slightly different approach that trains the algorithm agent against a mixture of hard instances and some randomly generated instances. See Appendix D for details.

Other related work. Recently, there is another line of work designing online algorithms with predictions given by machine learning (Antoniadis et al., 2020; Wang et al., 2020; Diakonikolas et al., 2021; Purohit et al., 2018; Wei & Zhang, 2020; Li et al., 2023a;b; Yang et al., 2023), with an objective to see how much improvements can be made on competitive ratios with such predictions. Although they also concern about theoretical guarantees, their bounds depend on how precise these predictions are. Our work focuses on a better understanding of challenges from the online optimization problem itself.

1.2. Roadmap

In Section 2, we present the formal definition of the OMSR problem, along with a benchmark problem we evaluate algorithms against, and provide an overview of existing algorithms and (upper and lower) bounds. We omit details that are not so necessary to grasp our main ideas and contributions for readers who are new to OMSR, or even online algorithms. Our framework, as illustrated in Figure 1, is described in detail in Section 3 and Section 4. Specifically, our framework is made up of two RL agents: an `adv` to generate hard (or worst-case) instances, and an `alg` to learn robust algorithms. Each agent views the other as its environment, and rewards are opposite to each other. We train the agents iteratively and from scratch: at first, `adv` is trained against an arbitrary or a simple known algorithm, for example, a greedy policy; `alg` is next learned against those hard instances; followed by repeating until a convergence. In our task on OMSR, we start from our baseline algorithm, Balance. After the iterative training, the hard instances learned by `adv` help give a provable upper bound for OMSR, beating the best-known upper bound in previous literature, as presented in Section 3. Section 4 will then present what `alg` learns against those hard instances, beating our baseline empirically.

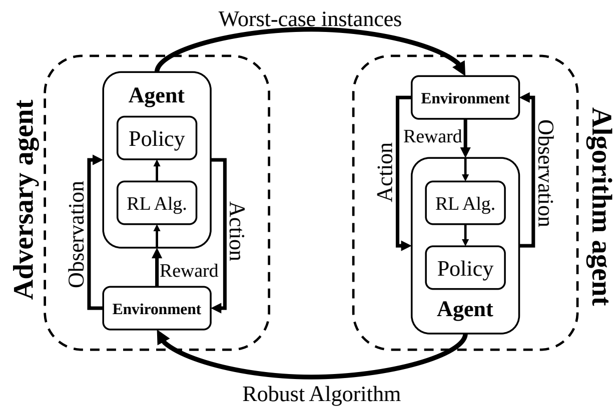


Figure 1. Our adversarial reinforcement learning framework.

2. Preliminaries

Formal definition of OMSR. Consider a bipartite graph $G = (U \cup V, E)$. There are n advertisers in $U = \{u_1, u_2, \cdots, u_n\}$ known in advance, and m search requests in $V = (v_1, v_2, \cdots, v_m)$ arriving one by one. Upon the arrival of each search request $v_j, j \in [m]$, its adjacent edges are revealed associated with a vector $p_j = (p_{1j}, p_{2j}, \cdots, p_{nj})$ to represent the success probabilities (click-through-rates) on every edge [4]. The algorithm then has to choose an available u_i and assign v_j to it immediately and irreversibly. If v_j is assigned to u_i, a coin is tossed to determine whether or not this assignment is *successful* with probability p_{ij}, independent of previous outcomes. If so, u_i becomes unavailable for future online vertices.[5] Conversely, if the assignment is *unsuccessful*, u_i remains available for another attempt of assignment. The objective of OMSR is to maximize the *expected number of successful matchings*. Existing studies vary on: (1) assuming success probabilities are sufficiently small [6] (e.g. $p_{ij} < 0.01, \forall (i,j) \in E$) or not; (2) assuming success probabilities are identical (i.e., $\forall (i,j) \in E$, $p_{ij} = p$) or not. For a clear presentation for

[4] If their is no edge between some u_i and v_j, let $p_{ij} = 0$.

[5] Note that in classic OBM, there are no stochastic processes on edges, so any assignments made by algorithms are successful in the context of this problem.

[6] OMSR is simplified under this assumption.

readers not familiar with OMSR, we focus on the restricted case of identical and infinitesimal success probabilities in our paper, but our experiments are not restricted to this assumption in Section 4. Under this assumption, the input instance is simplified as a matrix $\mathcal{P}_{n\times m}$ where for each $(i,j)\in E$, $p_{ij}=p$, and otherwise $p_{ij}=0$.

Benchmark and evaluation. To evaluate the performance of an algorithm, denoted as $\texttt{ALG}$, we compare it to an offline and non-stochastic version of OMSR, named *budgeted allocation* (BA) problem (Mehta & Panigrahi, 2012; Huang & Zhang, 2020) (See Appendix A for details). For a given instance (input graph) G, we define the *optimal solution*, denoted as $\texttt{OPT}(G)$, as the maximum objective value can be found over the same G of BA. Further, we define the *competitive ratio* ($\texttt{CR}$) of an algorithm as the infimum of ratio between $\texttt{ALG}(G)$ and $\texttt{OPT}(G)$ over all possible input graphs in an input space $\mathcal{G}$, that is, $\texttt{CR}=\inf_{G\in\mathcal{G}}\frac{\texttt{ALG}(G)}{\texttt{OPT}(G)}$. Note that $\texttt{CR}\in[0,1)$, and the closer to 1 the better. We make an early clarification on our abuse of $\texttt{CR}$ in our experiments: $\texttt{CR}$ is defined as the competitive ratio of an algorithm on a specific input graph G, i.e., $\texttt{CR}=\frac{\texttt{ALG}(G)}{\texttt{OPT}(G)}$.

Existing algorithms and bounds. For hardness result (upper bound), Mehta & Panigrahi (2012) prove that *no* algorithm for OMSR has a competitive ratio of more than $0.621<1-1/e\approx 0.632$, indicating that OMSR is strictly harder than OBM, for which the optimal $\texttt{CR}$ is 0.632. As one of our main results, we improve this bound to 0.597. For algorithm design (lower bound), the current SOTA is achieved by an algorithm named Balance (Huang & Zhang, 2020), which is also our baseline for evaluating learned algorithms. Balance [7] is a deterministic greedy strategy presented as follows and it is our starting point for training adv as the environment at the first iteration. $\texttt{CR}$ of Balance varies on whether probabilities on all edges are equal. Table 1 concludes the above results related to our work.

Balance Algorithm:

> Upon the arrival of each online vertex, match it to a neighbor with the *fewest* failure attempts.

3. Learning Worst Cases: a Provable Better Bound

Recall that our adversarial RL framework trains adv and alg iteratively. This section first presents how to set up the adv agent to learn hard instances against the current alg in every single iteration. At the very beginning, we could initialize alg as any random policy. In our experiments, to compare with the baseline, we train adv to generate worst-case instances against Balance in the first iteration. After several iterations such that the agents converge to a Nash equilibrium, graph instances can be sampled from the learning outcomes of adv in each iteration. We distill two properties related to learned patterns, which are what we call a *consistency property* and an *exclusivity property*, from statistical observations. Combining them up proves an improved upper bound for OMSR as stated in Table 1.

3.1. MDP Formulation

We formulate the adv as a Markov Decision Process (MDP) model. Recall that in OMSR, the input instance is a matrix $\mathcal{P}_{n\times m}$ for n offline vertices and m online vertices. $p_{ij}=p$ or 0 for $\forall i\in n.j\in m$ according to whether there exists an edge (i,j). Columns are revealed online one by one. In a nutshell, adv is trained to construct $\mathcal{P}$ gradually, generating the j-th column $\mathcal{P}_{[:,j]}$ of $\mathcal{P}$ at the state s_j. alg serves as the environment, which runs the current algorithm on $\mathcal{P}$ and produces a reward equal to $1-\texttt{CR}$. Details are presented as follows.

Environment. The current matching policy is defined by alg, and a calculator on $\texttt{CR}$.

State Space. A state s_j at a timestep j is the current (partial) matrix of $\mathcal{P}$, i.e. $\mathcal{P}_{[:,1:j-1]}$. A terminal state $\hat{S}$ is reached when a complete $\mathcal{P}$ is generated. The length of an episode is $T=m$.

Action space. At state s_j, an action $a_j\in A_j$ taken by adv is to select a subset $U'\subseteq U$, and set $p_{ij}=p$ for $i\in U'$, in other words, determine the neighbor of v_j. The size of action space $|A_j|$ is 2^n.

Transition. The transition from state s_j to the next state s_{j+1} is *deterministic*. If action a_j is taken, the j-th column in matrix $\mathcal{P}$ is updated and moves on to the next episode step, $j+1$.

Reward. If an episode ends, all taken actions will receive a reward of $1-\texttt{CR}$, where $\texttt{CR}$ is the competitive ratio of the current algorithm environment running on the generated instance.

Policy. At state s_j, a stochastic policy $\pi(a_j|s_j)$ outputs a distribution, the support of which is 2^U.

3.2. Training Algorithm

We use the Cross-Entropy method for reinforcement learning. The steps are as follows with hyperparameters $N_{\text{batch}}=1024$ and $\alpha=30\%$:

1. Initialize the policy distribution $\pi^{(0)}$ as random.
2. Generate N_{batch} matrices of $\mathcal{P}$ from $\pi^{(k)}$.
3. For each episode, compute the total reward by summing up all the rewards at each step.

[7]Balance is for the special case when all success probabilities equal. See a generalized-Balance algorithm in Appendix B.2 for the case without this assumption.

Table 1. Existing bounds of OMSR

	Lower Bound (Algorithms)	Upper Bound (Hardness)
Equal Prob.	0.576 (Balance) (Huang & Zhang, 2020)	0.621 (Mehta & Panigrahi, 2012) → 0.597 (This paper)
Unequal Prob.	0.572 (Huang & Zhang, 2020)	

4. Select the top α elite episodes with the highest total rewards.
5. Update the policy to $\pi^{(k+1)}$ for the set S of steps (s, a, r) in elite episodes by:

$$\pi^{(k+1)} = \arg\max_{\pi} \sum_{(s,a,r)\in S} \log \pi(a|s). \quad (1)$$

6. Repeat steps 2-5 until convergence.

We use a feed-forward neural network with a single hidden layer with 2^{n+2} neurons and ReLU for non-linearity. A fixed learning rate for the Adam optimizer is set as 10^{-3}. We also take an ϵ-greedy strategy, where the agent takes the action given by π with probability ϵ, and takes a random action with probability $(1 - \epsilon)$. We set $\epsilon = 0.5$.

3.3. Bridging Statistics and Theory

To understand in depth what adv learns and how it behaves in generating worst cases, we conduct experiments towards the output of adv's networks in every iteration. Our empirical observations indicate two immediate properties on the learned graph patterns, which contribute to a formal proof of the upper bound as claimed. Recall that the size of graph instances is $n \times m$, where n and m denote the number of offline and online vertices, respectively. We experiment on different choices of n and m. Besides, we also consider the success probability p varying from 0 to 1.

Expected number of offline neighbors. For each online vertex, adv outputs a distribution on its offline neighbor subset. Figure 2 plots the expected number of neighbors for online vertices from 1 to m in their arriving order. The larger numbered vertices come later. We experiment on $m = 240$ (Figure 2(a)) and $m = 200$ (Figure 2(b)). We observe a clear trend that *vertices arrive later may have fewer neighbors*. Such observations are in fact consistent with our intuition in constructing hard instances in OBM and its related problems: Early matching mistakes will lead to few choices for later arriving vertices, which are constructed with fewer neighbors.

Online vertices with identical neighbors. Another key observation is the adjacent arriving online vertices tend to share identical neighbors. To see this, we experiment on graph instances with only two offline vertices, that is, $n = 2$.

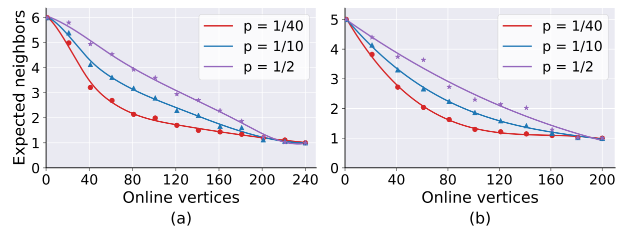


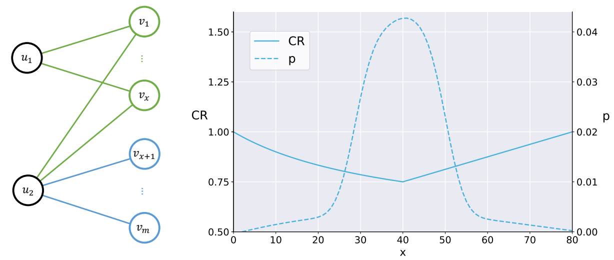
Figure 2. The expected number of each online vertex neighbors. The curve plots the expectation as a function of the index of online vertex.

Figure 3(a) shows instances sampled from adv's output. In this case, the first x vertices have edges with both offline vertices, while the remaining ones has only one neighbor. To further validate such finding, we restrict adv generates instances as Figure 3(a) and train it to find an optimal number of x. Figure 3(b) shows the probability distribution of x and the competitive ratios on the corresponding graph instances. The competitive ratio is minimized when $x = 40$, meaning that the worst instance is conducted when the first half online vertices has two neighbors, while the other half has only one.

To summarize the above empirical findings, we define a *consistency property* as follows, which restricts the graph patterns that adv generates, and thus reduces the size of the action space.

Definition 3.1 (Consistency Property)**.** A bipartite graph G satisfies a consistency property if the online vertices V of G can be partitioned into k disjoint subsets $V_1, V_2, \ldots, V_k$, such that $V = V_1 \cup V_2 \cup \cdots \cup V_n$, and for each $i \in [k]$, all vertices in V_i have the same neighbors.

Correlations between online vertices groups. Consistency property allows us to partition the online vertices into groups such that vertices in the same group can be viewed identically. So in the following discussion, we use a single online vertex to represent a group of online vertices. We further investigate correlations on how these groups are connected to offline neighbors. Take $n = 4$ as an example. Figure 4 presents hard instances sampled from adv's output with the smallest four competition ratios during all training iterations until the convergence. We define an *exclusivity property* as follows to capture the learned patterns. Intu-



(a) (b)

Figure 3. Figure (a) presents the worst-case instances learned under $n = 2$, where the first x (green) vertices have both neighbors, and the remaining $m - x$ (blue) vertices have only one neighbor. The curves in Figure (b) plot both the CR as a function of the value x, and the probability distribution p of x.

itively, the exclusivity property ensures the later coming online vertex can only have edges to offline vertices, which have *identical* earlier coming online neighbors. For example, in the last graph of Figure 4, vertex green connects to all four offline vertices. Vertex blue connects to the first three, and vertex orange connects to a subset of blue's neighbors (also a subset of green's neighbors). Definition 3.2 gives a formal definition.

Definition 3.2 (Exclusivity Property)**.** Given a bipartite graph G with online vertices V. Let N_j denote the neighbor set of a vertex $v_j \in V$. G is said to satisfy an exclusivity property if for each $j \in [m]$ and any $j' < j$, $N_j \subseteq N_{j'}$ or $N_j \cap N_{j'} = \varnothing$.

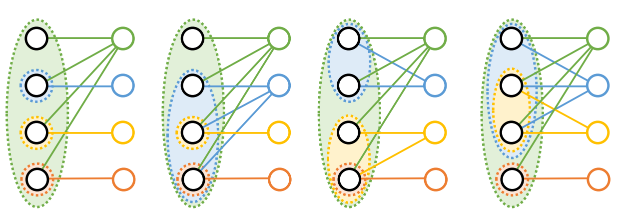


Figure 4. Worst-case instances learned under $n = 4$. Each instance has four offline vertices drawn on the left-hand-side, and four groups of online vertices drawn on the right-hand-side. Each online vertex group is drawn in a different color, and its neighbors are circled in the same color.

3.4. A Provable Upper Bound for OMSR

The properties introduced above not only effectively reduce the action spaces for training adv, but also contribute to a useful lemma that suffices to prove our main theoretical results as stated in Theorem 3.3.

Theorem 3.3. *No algorithm for the OMSR problem can achieve a competitive ratio of more than* 0.597.

Proof sketch. We provide a proof sketch of Theorem 3.3 in the rest of this section, and leave a formal proof in Appendix C. The proof consists of two ingredients: we first show that the Balance algorithm is optimal on a family of instances that satisfies both consistency and exclusivity properties as defined in Definition 3.1 and Definition 3.2. By such optimality, to prove an upper bound γ of OMSR, it then suffices to find instances where Balance is γ-competitive.

Consider the graph instances learned by adv as introduced in earlier sections. For a ease of notation, let $\mathcal{G}_n$ denote graph instances with n offline vertices and satisfies both consistency and exclusivity properties as defined in Definition 3.1 and Definition 3.2. Further, or easy calculation on $\texttt{OPT}(\mathcal{G}_n)$ [8], we restrict all vertices groups in $\mathcal{G}_n$ contains $1/p$ identical online vertices. $\mathcal{G}_n$ is formally defined as:

Definition 3.4. $\mathcal{G}_n$ is a family of bipartite graphs, which satisfy the consistency and exclusivity properties, with offline vertices $U = \{u_1, u_2, \cdots, u_n\}$ and online vertices $V = V_1 \cup V_2 \cup \cdots \cup V_n$. For each $i \in n$, V_i contains $1/p$ identical vertices that have the same probabilities vector towards U. Besides, probabilities on edges between u_i and all vertices in V_i can not be 0, i.e., $\forall v_j \in V_i$, $p_{ij} = p$.

Remarks on $\mathcal{G}_n$. By definition, $\mathcal{G}_n$ has n offline vertices and n/p online vertices divided into n groups. Further, for $\forall n \in Z_+$, $\texttt{OPT}(\mathcal{G}_n) = n$. Much more importantly, $\mathcal{G}_n$ brings great convenience for a theoretical proof on upper bound through the following lemma.

Lemma 3.5. *Balance is optimal on* $\mathcal{G}_n$*.*

Lemma 3.5 indicates the optimality of Balance on $\mathcal{G}_n$. To show an upper bound of 0.597 of OMSR, it then suffices to construct instances from $\mathcal{G}_n$ where Balance is 0.597-competitive. For a consideration of the page limit and a clear presentation, we provide proof of a weaker bound of 0.61 on $\mathcal{G}_3$, which is slightly larger than 0.597 as claimed in Theorem 3.3, but yet beats the previous best-known 0.621 as stated by Mehta & Panigrahi (2012).

Lemma 3.6. *There exists a subset of instances in* $\mathcal{G}_3$*, such that no algorithm on them can achieve a competitive ratio of more than* 0.61.

Let's see how Balance runs on one of these instances $G_3^\star \in \mathcal{G}_3$ as presented in Figure 5 and compute the expected number of successful offline vertices. Briefly speaking, we need to compute the probabilities of whether u_1, u_2, and u_3 are successful, together with 2^3 cases, and take an expectation. The expected number of successful vertices in u_1, u_2, u_3 is $3(1 - 11/(18e) - 11/(9e^2))$, deriving a competitive ratio of $(1 - 11/(18e) - 11/(9e^2)) < 0.61$. The complete proof is shown in Appendix C.

[8] Computing the optimal solution of an offline BA problem need run a Max-Flow algorithm.

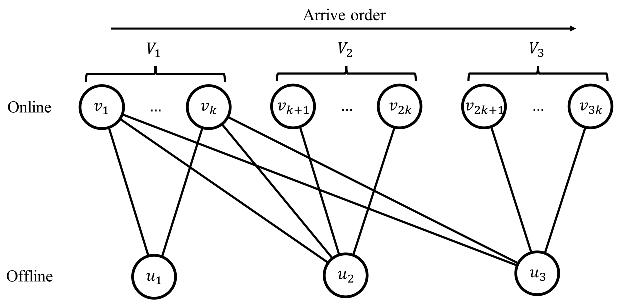


Figure 5. $G_3^\star$ with $|U| = 3$ and $|V| = 3k$, where $k = 1/p$ and $p \to 0$. Online vertices are listed at the top and arrive from left to right. Offline vertices are at the bottom.

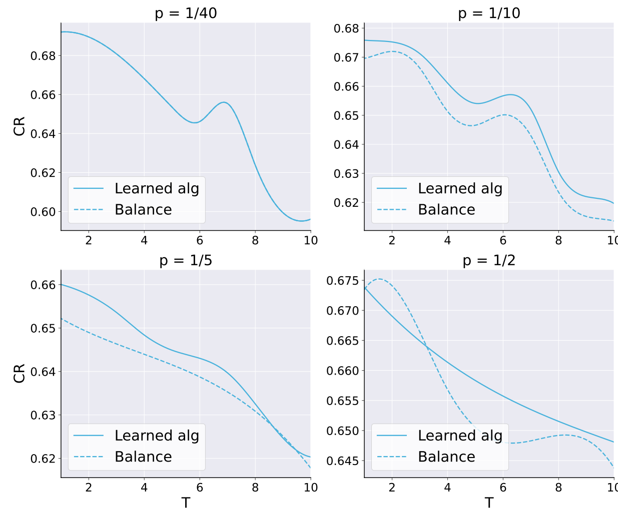


Figure 6. The average CRs of learned algorithm and Balance running on the learned worst-case distributions in each iteration T from 1 to 10. There are 4 experiments on different $p = 1/40, 1/10, 1/5$ and $1/2$. The solid and dashed curves plot the CR as a function of the iteration T for learned algorithm and Balance, respectively. In the figure with $p = 1/40$, the solid and dashed curves are nearly coincident.

Combining with Lemma 3.5 completes the proof of Lemma 3.6.

Finally, to obtain our final result as stated in Theorem 3.3, the adv finds $\mathcal{G}_n$'s as n grows up. The smallest competitive ratio, 0.597, is achieved when $n = 7$. Table 2 records these results as n grows from 1 to 10.

3.5. Evaluations on Hard Instances

We set up experiments to further validate the hardness of graph instances we construct.

The performance of algorithms on the learned worst-case distributions. To validate that adv can efficiently find hard instances in iterative training, we evaluate the performance of the algorithm learned by alg and Balance on the worst-case distributions in each iteration. We take experiments with different success probabilities ($p = 1/40, 1/10, 1/5$ and $1/2$), and plot the results in Figure 6. The results show that, in general, the performance of algorithms gets worse on the hard instances generated by the learned distributions in iterative training. In addition, we find that when the success probabilities are small enough (e.g. $p = 1/40$), the learned algorithm and Balance perform almost identically in the experiments. This may imply that Balance is optimal when p is vanishing.

$\mathcal{G}_n$ v.s. baselines. We compare the hardness of $\mathcal{G}_n$ to some other instances that are well known to be hard for OBM and its related problem, for example, the Thick-z and Triangular graphs (formally defined in Appendix B.1). Besides, we also compare with the worst-case instance constructed in Mehta & Panigrahi (2012). We experiment on the performances of algorithms learned by alg in each iteration from 0 to 10 running on these instances. Figure 7 shows that the learned algorithm gets better performance after iterative training, and converges to the corresponding CRs of these instances. The algorithm performs worst on the hard instances in our paper, and this experimentally indicates the generated instances are harder than other known worst-case instances.

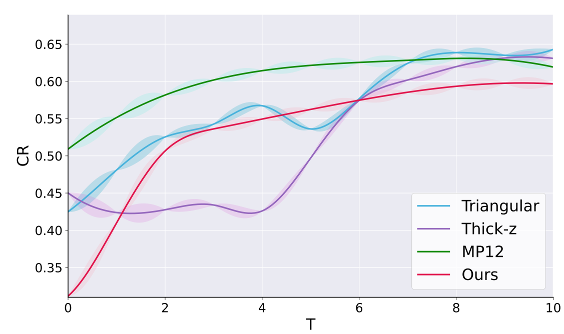


Figure 7. The average CRs of learned algorithms from iteration 0 to 10 running on some specific worst-case instances. The curve plots the CR as a function of the iteration T.

4. Learning Robust Algorithms: Experimental Evaluations

This section introduces the other component of our framework, the algorithm agent alg. We utilize a similar design for the agent as previous literature (Alomrani et al., 2021; Kong et al., 2019). The algorithm agent takes the input instances generated from adv in Section 3 as the environment, and learns robust algorithms under the given input instances.

Table 2. The competitive ratio on $\mathcal{G}_n$ rounded to three decimal places.

$\mathcal{G}_n$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n=10$
CR	0.632	0.623	0.610	0.605	0.604	0.599	**0.597**	0.597	0.598	0.599

We defer the MDP formulation and training details in Appendix D.

4.1. Evaluations

We evaluate the performances of algorithms learned by `alg` in both equal and unequal probabilities cases [9] by the following experiments. For an easy calculation on `OPT`, we restrict input instances to $\mathcal{G}_n$ as defined in Definition 3.4.

The average `CR`s of algorithms under worst-case distributions in different iterations. Let T denote the number of iterations. (An iteration consists of one training for `adv` and `alg`) We fix the number of offline vertices as $n = 6$, and the number of online vertices in one group is $1/p$, where p is the probability on the edges. For equal probability case, the probability on each edge is identical and sampled randomly. For unequal probability case, the probability on each edge is sampled independently. Figure 8 plots the average `CR` under worst-case distributions in iterations from 0 to 10. At the beginning ($T = 0$) when it is initialized as random, `alg` performs poorly against the trained adversary inputs, with around 0.307 (equal) and 0.225 (unequal). Average `CR`s converge to 0.635 (equal) and 0.614 (unequal), better than that of Balance.

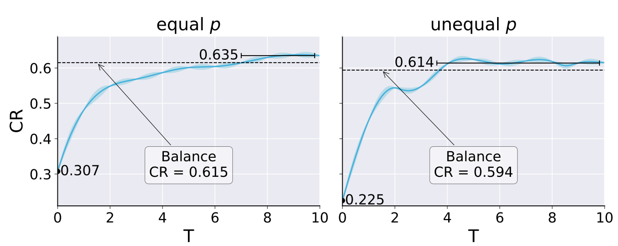


Figure 8. The average `CR`s of algorithms under worst-case distributions in iteration 0 to 10. The curve plots the `CR` as a function of the iteration T. As a comparison, the dashed line represents the `CR` of Balance under worst-case distributions in the last iteration (0.615 for equal and 0.594 for unequal).

The average `CR`s of algorithms with different sizes of offline vertices. Table 3 lists the average `CR`s of algorithms under different numbers of offline vertices n. The `CR`s of the learned algorithm and Balance are equal when $n = 1$. As n grows up, `CR`s of both algorithms decrease. Moreover, it can be observed that the learned algorithm consistently outperforms Balance on average.

[9] Please refer to Appendix D.3 for training details for the unequal probability case.

Table 3. The average `CR`s of algorithms under worst-case distributions with different numbers of offline vertices.

Case	Algotrithm	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
Equal	Learned alg	0.716	0.655	0.649	0.639	0.637	0.635
	Balance	0.716	0.654	0.649	0.635	0.621	0.615
Unequal	Learned alg	0.678	0.658	0.648	0.639	0.632	0.614
	Balance	0.678	0.622	0.617	0.610	0.600	0.594

The average `CR`s of algorithms under worst-case distributions with different success probabilities. For equal probability p, we train `adv` and `alg` with $m = n/p, n = 6$. Success probabilities p vary from 0 to 1. We observe in Figure 9 a tiny gap of `CR` when $p < 0.5$. When $p > 0.5$, our learned algorithms significantly outperform Balance.

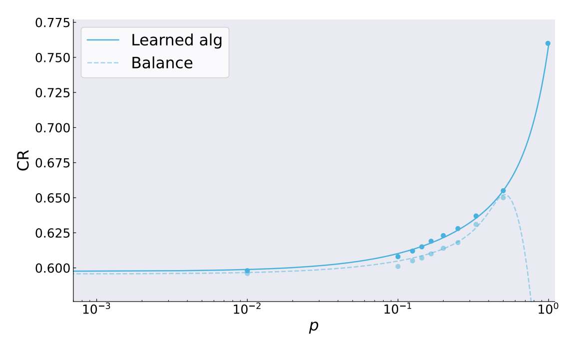


Figure 9. The average `CR`s of algorithms under worst-case distributions with $n = 6$ varying on p. As a comparison, the dashed line represents the `CR` of Balance.

5. Conclusions and Limitations

There has been extensive literature research on using ML methods to solve classic combinatorial optimization problems or apply techniques in combinatorial algorithms to enhance the ability of ML. However, research that uses ML to illuminate the understanding of online optimization problems is rare. To the best of our knowledge, this paper gives the first successful attempt by improving the best-known upper bound with insights from RL for a specific online matching problem. Besides, our framework can also learn robust (optimal) algorithms as done in previous literature. We point out a major limitation but also an interesting future research direction: while we get insights from what `adv` learns, it is pretty hard to summarize what `alg` learns although it performs better than the SOTA algorithm and its competitive ratio. If we could achieve it, the lower bound for OMSR would also be improved. Nevertheless, we believe that our framework gives a representative example and could be generalized to other online optimization problems.

Impact Statement

This paper advances the field of ML by enhancing the theoretical understanding of an online combinatorial problem via RL. To the best of our knowledge, this gives the first successful attempt in such a field. We believe in the great potential of ML to assist in theory studies in the future. In ethical aspects, we are not aware of related issues in this paper.

Acknowledgements

Qiankun Zhang is supported by National Natural Science Foundation of China (Grant No. 62302183) and CCF-Huawei Huyanglin Fund - Theoretical Computer Science Special Project (No. CCF-HuaweiLK2023004). Bingqian Du is supported by National Natural Science Foundation of China (Grant No. 62302187).

References

Alomrani, M. A., Moravej, R., and Khalil, E. B. Deep policies for online bipartite matching: a reinforcement learning approach. *arXiv preprint arXiv:2109.10380*, 2021.

Antoniadis, A. F., Gouleakis, T., Kleer, P., and Kolev, P. Secretary and online matching problems with machine learned advice. *ArXiv*, abs/2006.01026, 2020.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *Eur. J. Oper. Res.*, 290:405–421, 2018.

Boutilier, C. and Lu, T. Budget allocation using weakly coupled, constrained markov decision processes. In Ihler, A. and Janzing, D. (eds.), *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA*. AUAI Press, 2016.

Diakonikolas, I., Kontonis, V., Tzamos, C., Vakilian, A., and Zarifis, N. Learning online algorithms with distributional advice. In *International Conference on Machine Learning*, 2021.

Dong, J., Hsu, H.-L., Gao, Q., Tarokh, V., and Pajic, M. Robust reinforcement learning through efficient adversarial herding. *arXiv preprint arXiv:2306.07408*, 2023.

Du, B., Huang, Z., and Wu, C. Adversarial deep learning for online resource allocation. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 6(4):1–25, 2022a.

Du, Y., Li, J., Li, C., and Duan, P. A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, 2022b.

Gisslén, L., Eakins, A., Gordillo, C., Bergdahl, J., and Tollmar, K. Adversarial reinforcement learning for procedural content generation. In *2021 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2021.

Goyal, V. and Udwani, R. Online matching with stochastic rewards: Optimal competitive ratio via path based formulation. *Proceedings of the 21st ACM Conference on Economics and Computation*, 2019.

Hua, X., Wang, X., Rui, T., Shao, F., and Wang, D. Adversarial reinforcement learning with object-scene relational graph for video captioning. *IEEE Transactions on Image Processing*, 31:2004–2016, 2022.

Huang, Z. and Zhang, Q. Online primal dual meets online matching with stochastic rewards: configuration lp to the rescue. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1153–1164, 2020.

Huang, Z., Zhang, Q., and Zhang, Y. Adwords in a panorama. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1416–1426. IEEE, 2020.

Huang, Z., Jiang, H., Shen, A., Song, J., Wu, Z., and Zhang, Q. Online matching with stochastic rewards: Advanced analyses using configuration linear programs. In *International Conference on Web and Internet Economics*, pp. 384–401. Springer, 2023.

Jiang, Y., Zhang, T., Ho, D., Bai, Y., Liu, C. K., Levine, S., and Tan, J. Simgan: Hybrid simulator identification for domain adaptation via adversarial reinforcement learning. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2884–2890, 2021.

Karp, R. M., Vazirani, U. V., and Vazirani, V. V. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 352–358. ACM, 1990.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Kong, W., Liaw, C., Mehta, A., and Sivakumar, D. A new dog learns old tricks: Rl finds classic optimization algorithms. In *International conference on learning representations*, 2019.

Kool, W., Van Hoof, H., and Welling, M. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Li, P., Yang, J., and Ren, S. Learning for edge-weighted online bipartite matching with robustness guarantees. *arXiv preprint arXiv:2306.00172*, 2023a.

Li, P., Yang, J., Wierman, A., and Ren, S. Robust learning for smoothed online convex optimization with feedback delay. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Ma, X., Driggs-Campbell, K., and Kochenderfer, M. J. Improved robustness and safety for autonomous vehicle control with adversarial reinforcement learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1665–1671. IEEE, 2018.

Mazyavkina, N., Sviridov, S. V., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.*, 134:105400, 2020.

Mehta, A. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

Mehta, A. and Panigrahi, D. Online matching with stochastic rewards. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pp. 728–737. IEEE, 2012.

Mehta, A., Saberi, A., Vazirani, U. V., and Vazirani, V. V. Adwords and generalized on-line matching. *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pp. 264–273, 2005.

Mehta, A., Waggoner, B., and Zadimoghaddam, M. Online stochastic matching with unequal probabilities. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1388–1404. SIAM, 2014.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. K. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, 2017.

Purohit, M., Svitkina, Z., and Kumar, R. Improving online algorithms via ml predictions. In *Neural Information Processing Systems*, 2018.

Shao, Z., Yang, J., Shen, C., and Ren, S. Learning for robust combinatorial optimization: Algorithm and application. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pp. 930–939. IEEE, 2022.

Spooner, T. and Savani, R. Robust market making via adversarial reinforcement learning. *arXiv preprint arXiv:2003.01820*, 2020.

Vinitsky, E., Du, Y., Parvate, K., Jang, K., Abbeel, P., and Bayen, A. Robust reinforcement learning using adversarial populations. *arXiv preprint arXiv:2008.01825*, 2020.

Wang, S.-F., Li, J., and Wang, S. Online algorithms for multi-shop ski rental with machine learned advice. *arXiv: Data Structures and Algorithms*, 2020.

Wang, Y., Tong, Y., Long, C., Xu, P., Xu, K., and Lv, W. Adaptive dynamic bipartite graph matching: A reinforcement learning approach. In *2019 IEEE 35th international conference on data engineering (ICDE)*, pp. 1478–1489. IEEE, 2019.

Wei, A. and Zhang, F. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. *Advances in Neural Information Processing Systems*, 33: 8042–8053, 2020.

Wu, T., Zhong, F., Geng, Y., Wang, H., Zhu, Y., Wang, Y., and Dong, H. Grasparl: Dynamic grasping via adversarial reinforcement learning. *ArXiv*, abs/2203.02119, 2022.

Yang, J., Li, P., Li, T., Wierman, A., and Ren, S. Anytime-competitive reinforcement learning with policy prior. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Yang, Y. and Whinston, A. A survey on reinforcement learning for combinatorial optimization. *ArXiv*, abs/2008.12248, 2020.

Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S., and Chi, X. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020a.

Zhang, K., Hu, B., and Basar, T. On the stability and convergence of robust adversarial reinforcement learning: A case study on linear quadratic systems. *Advances in Neural Information Processing Systems*, 33:22056–22068, 2020b.

Zuzic, G., Wang, D., Mehta, A., and Sivakumar, D. Learning robust algorithms for online allocation problems using adversarial training. *arXiv preprint arXiv:2010.08418*, 2020.

A. Benchmark: Budgeted Allocation (BA) Problem

Formal definition of BA. Consider a bipartite graph $G(U \cup V, E)$ with n vertices in U and m vertices in v *all known* at the very beginning. For each $i \in [n]$ and each $j \in [m]$, edge (u_i, v_j) is associated with p_{ij}. Compared to OMSR, p_{ij} represents a *weight* instead of a probability over an edge. For each vertex v_j, the algorithm has to assign it to one of its neighbors u_i, and after that, u_i gains a deterministic weight of p_{ij}. The total weights gained by each u_i are constrained to be no greater than 1. The objective of this problem is to maximize the total weights among all vertices in U.

We address that BA is a proper benchmark for OMSR, basically because for any given instances, the expected solution of OMSR can be upper bounded by the optimal solution of BA. Readers may refer to Lemma 1 in Mehta & Panigrahi (2012) for details.

B. Baselines

B.1. Baselines for Hard Instances

Triangular graphs. If an instance $G = (U \cup V, E)$ is a triangular graph, it satisfies that $U = \{u_1, u_2, \cdots, u_n\}$, $V = V_1 \cup V_2 \cup \cdots \cup V_n$, and the neighbors of each online vertex in set V_i is $\{u_i, u_{i+1}, \cdots, u_n\}$. The size of V_i is randomly chosen from the set $\{k_1, k_1 + 1, \cdots, k_2\}$, where k_1 and k_2 are hyperparameters. The probabilities on edges associated with vertex in V_i are $1/|V_i|$.

Thick-z graphs. If an instance $G = (U \cup V, E)$ is a thick-z graph, it satisfies that $U = \{u_1, u_2, \cdots, u_{2n}\}$, $V = V_1 \cup V_2 \cup \cdots \cup V_{2n}$, the neighbors of each online vertex in the first n set $V_i, i \in [n]$ is $\{u_i\} \cup \{u_{n+1}, u_{n+2}, \cdots, u_{2n}\}$, and the vertex in the last n set $V_i, i \in (n, 2n]$ has only one neighbor u_i. The size of V_i is randomly chosen from the set $\{k_1, k_1 + 1, \cdots, k_2\}$, where k_1 and k_2 are hyperparameters. The probabilities on edges associated with vertices in V_i are $1/|V_i|$.

Random graphs. We used a random graph generator to generate some random graphs. For a graph $G = (U \cup V, E)$ with $U = \{u_1, u_2, \cdots, u_{2n}\}$ and $V = V_1 \cup V_2 \cup \cdots \cup V_{2n}$, the generator first determines the edge set E. For each pair (u, V), the generator determines whether all edges (u, v), $v \in V$, should be included in E, with a probability of 0.5. The size of V_i is randomly chosen from the set $\{k_1, k_1 + 1, \cdots, k_2\}$, where k_1 and k_2 are hyperparameters. The probabilities on edges associated with vertices in V_i are $1/|V_i|$.

Hard-instances in MP12 (Mehta & Panigrahi, 2012). The hard-instances $G = (U \cup V, E)$ in MP12 satisfies that $U = \{u_1, u_2, \cdots, u_n\}$, $V = V_1 \cup V_2 \cup \cdots \cup V_n$, and the neighbors of each online vertex in set V_i is $\{u_i, u_{i+1}, \cdots, u_n\}$. Choose value k from the set $\{k_1, k_1 + 1, \cdots, k_2\}$, where k_1 and k_2 are hyperparameters, and set sizes of $V_i, i \in [n]$ as k. The probabilities on all edges are $1/k$.

B.2. Baselines for Robust Algorithms: Balance

Balance (Equal Probability). On the arrival of each online vertex $v_j \in V$, Balance matches it to the available neighbor with the least failure attempts, breaking ties arbitrarily. Huang & Zhang (2020) prove Balance achieves 0.576-competitive under the small-probability assumption.

Generalized-Balance (Unequal Probability). In this case, success probabilities vary on edges, meaning that each edge contributes differently to the marginal gains of the offline vertex. Instead of the number of failure attempts, Generalized-Balance uses a similar indicator, total success probabilities on edges matched to an offline vertex, named as *load* L_i of an offline vertex u_i. On the arrival of each online vertex $v_j \in V$, match it to the available neighbor $u_{i^\star}$ with $i^\star = \arg\max_{1 \le i \le |U|} p_{ij}(1 - g(L_i))$, where $g : [0, 1] \to [0, 1]$ is a monotonic non-decreasing function, and break ties arbitrarily. Readers may easily verify when all $p_{ij} = p$, it reduces to the algorithm in equal probabilities case. Generalized-Balance can achieve 0.572-competitive under the small-probability assumption.

C. Omitted proofs

C.1. Proof of Lemma 3.5

For each offline vertex $u \in U$, let the *load* L_u denote the total success probabilities on edges matched to u. The insight and details on the load definition are in Appendix B.2. L_u is capped by 1. To better capture the stochastic rewards in our analysis, we change the objective of this problem from maximizing the expected number of successful offline vertices to maximizing the total loads by Lemma C.1.

Lemma C.1. *For any algorithm running on $\mathcal{G}_n$, the expected load on a vertex u is equal to its probability of success.*

Proof. Let x_{uv} be the random variables that if the algorithm matches u to v, set $x_{uv} = 1$; and otherwise, set $x_{uv} = 0$. Recall all probabilities on the edges are equal to p and $p \to 0$. The expected load on u by definition is $\mathbb{E}[L_u] = \sum_{v \in V} p \cdot \mathbb{E}[x_{uv}]$. For a specific matching assignments $x_u = (x_{u1}, \cdots x_{um})$, the probability that u succeeds is $1 - \prod_{v \in V}(1 - p \cdot x_{uv}) = 1 - \exp(\sum_{v \in V} -p \cdot x_{uv}) = 1 - (-p \cdot \sum_{v \in V} x_{uv} + 1) = p \cdot \sum_{v \in V} x_{uv}$ $(p \to 0)$. Therefore, the probability that u succeeds under the matching assignments produced by algorithm is that $\mathbb{E}[p \cdot \sum_{v \in V} x_{uv}] = \sum_{v \in V} p \cdot \mathbb{E}[x_{uv}] = \mathbb{E}[L_u]$. □

The next two lemmas show the key property to prove the optimality and how Balance performs with this property on $\mathcal{G}_n$.

Let N_i denote the shared neighbors of vertices in V_i.

Lemma C.2. *There exists an optimal algorithm on $\mathcal{G}_n$ that equally distributes the load of each group V_i among all V_i's neighbors.*

Proof. Configure a set partition process. During the execution of the algorithm, it partitions the offline vertex set into a family of subsets $\mathbb{S}$. Initially, set $\mathbb{S} = \{U\}$. When the first group of online vertices V_1 arrives, $N_1 \subseteq U$. At this time, partition U to N_1 and $U \setminus N_1$, and replace U with non-empty set N_1 and $U \setminus N_1$ in $\mathbb{S}$. For the algorithm, since L_u for all u are equal to zero, these offline vertices are identical to the algorithm.

Suppose that algorithm (unequally) distributes the load increment $\ell_1, \cdots, \ell_{|N_1|}$ for each offline vertices, and $\ell_{\pi(1)} \leq \cdots \leq \ell_{\pi(|N_1|)}, \ell_{\pi(1)} < \ell_{\pi(|N_1|)}$, where π is a permutation of $|N_1|$. If this assignment is better than equal distribution, this means there exists at least one offline vertex that accommodates more load. W.l.o.g., assume that this vertex is $\pi(1)$. Let $\bar{\ell}$ denote the load increment when equally distributing the load, and inequality $\ell_{\pi(1)} < \bar{\ell} < \ell_{\pi(|N_1|)}$ holds. Thus, if $\pi(1)$ can accommodate more load, the load increment of $L_{\pi(1)}$ by the unrevealed arrivals is greater than $1 - \bar{\ell}$, since $L_{\pi(1)}$ is capped by 1. But the input is adversarial, imagine there is an adversary who reveals the online vertices according to the previous assignments made by the algorithm. Thus, the adversary can rearrange the order of offline vertices in N_1 for the unrevealed vertices, because these vertices in N_1 are identical. So, the input can be modified as swapping vertices $\pi(1)$ and $\pi(|V_1|)$. At this time, future increment of the load has to be $1 - \ell_{\pi(|N_1|)} < 1 - \bar{\ell}$. The result is worse than equally distributing the load. By contradiction, we show that equally distributing the load of V_i is optimal. Further, after the assignment of V_i, the vertices in N_1 have the same load $\bar{\ell}$.

When the i-th group of online vertices V_i arrives, by Definition 3.2, there exists $S \in \mathbb{S}$, such that $N_i \subseteq S$. It also partitions N_i to N_i and $S \setminus N_i$, and replace S with non-empty set N_i and $S \setminus N_i$ in $\mathbb{S}$. For the algorithm, the vertices in N_i all have the same load. Therefore, it is optimal for the algorithm to distribute the load equally based on the previous analysis. This completes the proof by induction on i. □

Lemma C.3. *When Balance runs on $\mathcal{G}_n$, it equally distributes the load of each group V_i among all V_i's neighbors.*

Proof. Configure the same set partition process in the proof of Lemma C.2. When the first group of online vertices V_i arrives, the loads of all the vertices are equal to 0. Since Balance matches each online vertices to the neighbor with the lowest load, and $|V_i| = 1/p, p \to 0$, it distributes the load from V_1 to all its neighbors equally. When the i-th group of online vertices V_i arrives, the set $S \in \mathbb{S}$, such that $N_i \subseteq S$, the vertices in N_i also has the same load, so Balance equally distributes the load of V_i. This completes the proof by induction on i. □

Lemma C.2 and Lemma C.3 complete the proof of Lemma 3.5.

C.2. Proof of Lemma 3.6

Recall $G_3^\star$ in Figure 5, and $G_3^\star$ satisfies the symmetry property.

We first calculate the expected number of successful matchings of the Balance algorithm on $G_3^\star$, i.e. $\texttt{ALG}(G_3^\star)$. Consider the instance $G_1^\star \in \mathcal{G}_1$ where there is only one offline vertex u_1 and one set of online vertices V_1 in which vertices are connected to u_1, with equal probability $p \to 0$. Let $k = 1/p$. In this case, each vertex in V_1 tries to get matched with u_1, so u_1 success with probability $1 - (1-p)^k = 1 - (1-1/k)^k = 1 - 1/e$ $(k \to \infty)$. Thus, the expected number of successes for $G_1^\star$ is $1 - 1/e$.

For $G_3^\star$, we calculate the expected number of successes in 8 cases corresponding to the outcomes of vertices in V_1. Let $U^\star$ be a subset of $\{u_1, u_2, u_3\}$, and denote the set of offline vertices that get matched to a vertex in V_1 successfully.

These 8 cases are listed below according to the vertices in set $U^\star$.

1. $|U^\star| = 0$. There is one subcase with no success in $\{u_1, u_2, u_3\}$, i.e. $U^\star = \varnothing$, and this happens with probability $(1-p)^k = 1/e$; in this case, the expected number of successes overall is the same as double that for $G_1^\star$, i.e. $2(1-1/e)$.

2. $|U^\star| = 1$. This case happens with probability $\binom{k}{1} \cdot p(1-p)^{k-1} = 1/e$. There are three subcases with equal probability:
   (a) $U^\star = \{u_1\}$. This happens with probability $1/(3e)$; in this case, the expected number of successes for V_2 and V_3 is the same as that for $G_1^\star$, i.e. $2(1-1/e)$; Therefore, the expected number of successes overall is $3 - 2/e$.
   (b) $U^\star = \{u_2\}$. This happens with probability $1/(3e)$; in this case, the expected number of successes for V_2 and V_3 is $1 - 1/e$, since no vertex in V_2 can be matched. Therefore, the expected number of successes overall is $2 - 1/e$.
   (c) $U^\star = \{u_3\}$. This case is similar to last subcase, so the expected number of successes overall is $2 - 1/e$.

3. $|U^\star| = 2$. This case happens with probability $\binom{k}{2} \cdot p^2(1-p)^{k-2} = 1/(2e)$. There are three subcases with equal probability:
   (a) $U^\star = \{u_1, u_2\}$. This happens with probability $1/(6e)$; in this case, the expected number of successes for V_2 and V_3 is $1 - 1/e$; Therefore, the expected number of successes overall is $3 - 1/e$.
   (b) $U^\star = \{u_1, u_3\}$. This case is the same as last subcase, so the expected number of successes overall is $3 - 1/e$.
   (c) $U^\star = \{u_2, u_3\}$. This happens with probability $1/(6e)$; in this case, the number of successes overall is 2.

4. $|U^\star| = 3$. In this case, $U^\star = \{u_1, u_2, u_3\}$ happens with probability $1 - 5/(2e)1 - 5/(2e)$; in this case, the number of successes overall is 3.

Combining the above, the expected number of successes overall is $3(1 - 11/(18e) - 11/(9e^2))$. Therefore, the competitive ratio of $G_3^\star$ is $1 - 11/(18e) - 11/(9e^2) < 0.61$. That is, Balance achieves 0.61-competitive on $G_3^\star$.

Finally, together with Lemma 3.5 where Balance is optimal on $G_3^\star$, the proof is completed.

C.3. Proof of Theorem 3.3

We illustrate the worst-case instances $G_n^\star, n \in \{4, 5, 6, 7\}$ corresponding to the values in Table 2 in Figure 10. By Lemma 3.5, it is sufficient to compute the competitive ratio of the Balance algorithm on graph $G_7^\star$, which is 0.597.

Because $\texttt{OPT}(G_7^\star) = 7$, the ratio of $G_7^\star$ is $\frac{1}{7}\texttt{ALG}(G_7^\star)$. We calculate the expected number of success matches on $G_7^\star$ (i.e. $\texttt{ALG}(G_7^\star)$) in the rest of this section. We use a dynamic programming approach for the calculation, rather than what we do in the proof of Lemma 3.6.

Let $H_i, i \in [7]$ denote the set of offline vertices, which successfully match the online vertices in V_i.

Thus,

$$\texttt{ALG}(G_7^\star) = \sum_{i=1}^{7} \mathbb{E}\left[|H_i|\right] = \mathbb{E}\left[\sum_{i=1}^{7} |H_i|\right]. \tag{2}$$

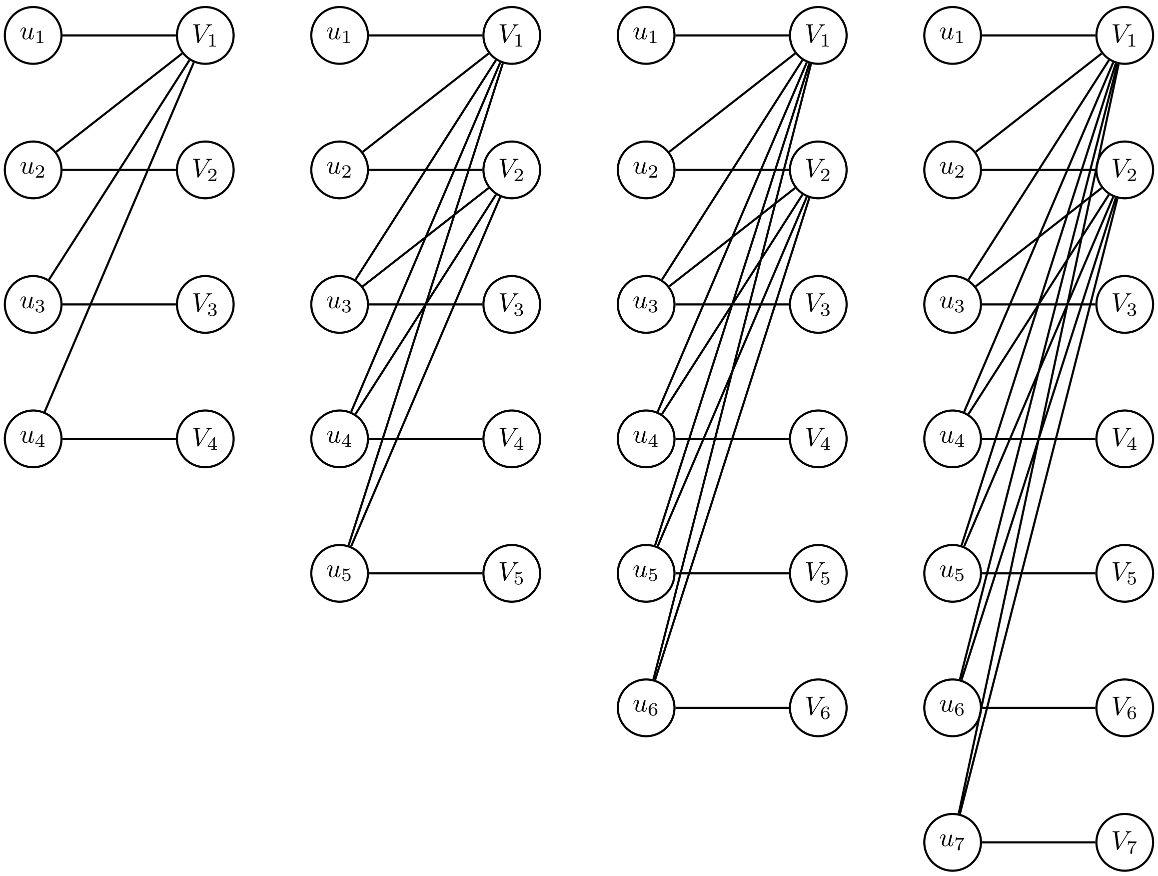


Figure 10. $G_n^\star, n = 4, 5, 6, 7$ from left to right with $|U| = n$ and $|V| = kn$, where $k = 1/p$ and $p \to 0$. To simplify the illustration, we use a single node to represent the online vertex set $V_i, i \in [n]$, with the edges representing the edges between all vertices in the online vertex sets and the offline vertices, since online vertices have the same neighbors.

As $H_i, i > 1$ depends on H_1, rephrase the expectation expression as a conditional expectation with respect to H_1:

$$\mathbb{E}\left[\sum_{i=1}^{7}|H_i|\right] = \sum_{h_1}\Pr[h_1]\mathbb{E}\left[\sum_{i=1}^{7}|H_i|\middle| H_1 = h_1\right] \tag{3}$$

$$= \sum_{h_1}\Pr[h_1]\left(|h_1| + \mathbb{E}\left[\sum_{i=2}^{7}|H_i|\middle| H_1 = h_1\right]\right). \tag{4}$$

$H_i, i > 2$ also depends on H_2, so rephrase the factor of $\mathbb{E}\left[\sum_{i=2}^{7}|H_i|\middle| H_1 = h_1\right]$ in Eqn.(4):

$$\mathbb{E}\left[\sum_{i=2}^{7}|H_i|\middle| H_1 = h_1\right] = \sum_{h_2}\Pr[h_2|H_1 = h_1]\mathbb{E}\left[\sum_{i=2}^{7}|H_i|\middle| H_1 = h_1, H_2 = h_2\right] \tag{5}$$

$$= \sum_{h_2}\Pr[h_2|H_1 = h_1]\left(|h_2| + \mathbb{E}\left[\sum_{i=3}^{7}|H_i|\middle| H_1 = h_1, H_2 = h_2\right]\right). \tag{6}$$

For Eqn.(6), the value of conditional expectation $\mathbb{E}\left[\sum_{i=3}^{7}|H_i|\middle| H_1 = h_1, H_2 = h_2\right]$ only depends on the number of remaining vertices in the set $\{u_3, u_4, \cdots, v_7\}$ after the occurrence of H_1 and H_2.

Thus, let $f_1(k)$ represent conditional expected value when there has k vertices remaining in the set $\{u_3, u_4, \cdots, v_7\}$ (after h_1 and h_2). Then we have

$$f_1(k) = k \cdot \texttt{ALG}(G_1^\star) = k\left(1 - \frac{1}{e}\right). \tag{7}$$

To calculate Eqn.(6), it needs to compute the value of $\Pr[h_2|H_1 = h_1]$. We define $p_i^{(n)}$ as follows, and recall that $k \to \infty$ is the size of V_i,

$$p_i^{(n)} = \begin{cases} \binom{k}{i}p^i(1-p)^{k-i} = \frac{1}{i!\cdot e} & i < n, \\ 1 - \sum_{i=0}^{n-1} p_i & i = n. \end{cases} \tag{8}$$

Therefore,

$$\Pr[h_2|H_1 = h_1] = \frac{p_{|h_2|}^{(7-|h_1|)}}{\binom{7-|h_1|}{h_2}}. \tag{9}$$

Let $f_2(j, k)$ represent the conditional expected value when u_2 is matched ($j = 0$) or unmatched ($j = 1$) and the set $\{u_3, u_4, \cdots, v_7\}$ has k vertices remaining after h_1. From Eqn.(7), (8) and (9), we rephrase Eqn.(6) using f_2

$$f_2(J, K) = \sum_{j=0}^{J}\sum_{k=0}^{K}\frac{p_{j+k}^{(J+K)}}{\binom{J+K}{j+k}} \cdot \binom{K}{k} \cdot (j + k + f_1(K - k)). \tag{10}$$

We calculate the values of $f_2(j, k)$ and present them in Table 4.

Table 4. The values of function $f_2(j, k)$ when $j = 0, 1$ and $k = 0, 1, \cdots, 5$.

$f_2(j,k)$	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$j=0$	0	$1-e^{-2}$	$2-3e^{-2}$	$3-\frac{11}{2e^2}$	$4-\frac{49}{6e^2}$	$5-\frac{87}{8e^2}$
$j=1$	$1-\frac{1}{e}$	$2-\frac{3}{2e}-\frac{3}{2e^2}$	$3-\frac{11}{6e}-\frac{11}{3e^2}$	$4-\frac{49}{24e}-\frac{49}{8e^2}$	$5-\frac{87}{40e}-\frac{87}{10e^2}$	$6-\frac{1631}{720e}-\frac{1631}{144e^2}$

Remarks. $\texttt{ALG}(G_3^\star) = f_2(1,2) = 3 - \frac{11}{6e} - \frac{11}{3e^2}$, and $\texttt{ALG}(G_4^\star) = f_2(1,3) = 4 - \frac{49}{24e} - \frac{49}{8e^2}$. Thus, the competitive ratio of Balance on $G_4^\star$ is $\frac{\texttt{ALG}(G_4^\star)}{\texttt{OPT}(G_4^\star)} = \frac{1}{4}(4 - \frac{49}{24e} - \frac{49}{8e^2}) \approx 0.605$.

Let $f_3(i,j,k)$ represent the conditional expected value when u_1 is matched ($i=0$) or unmatched ($i=1$) and u_2 is matched ($j=0$) or unmatched ($j=1$) and the set $\{u_3, u_4, \cdots, v_7\}$ has k vertices remaining. From Eqn.(4), we have

$$f_3(I,J,K) = \sum_{i=0}^{I}\sum_{j=0}^{J}\sum_{k=0}^{K} \frac{p_{i+j+k}^{(I+J+K)}}{\binom{I+J+K}{i+j+k}} \cdot \binom{K}{k} \cdot (i+j+k+f_2(J-j, K-k)). \tag{11}$$

We calculate the values of $f_3(i,j,k)$ and present part of them in Table 5.

Table 5. The values of function $f_3(i,j,k)$ when $i=j=1$ and $k=0,1,\cdots,5$.

k	$f_3(1,1,k)$
0	$2 - \frac{3}{2e} - \frac{3}{2e^2}$
1	$\frac{1}{6e^3}(-15 - 15e - 11e^2 + 18e^3)$
2	$\frac{1}{24e^3}(-164 - 82e - 49e^2 + 96e^3)$
3	$\frac{1}{40e^3}(-498 - 166e - 87e^2 + 200e^3)$
4	$\frac{1}{720e^3}(-13536 - 3384e - 1631e^2 + 4320e^3)$
5	$\frac{1}{5040e^3}(-128680 - 25736e - 11743e^2 + 35280e^3)$

The expected number of successes of $G_7^\star$ is $f_3(1,1,5)$. Therefore, Balance achieves competitive ratio of $\frac{1}{7}f_3(1,1,5) \approx 0.597$ on $G_7^\star$.

Remarks. Using the values of function f_3, we can obtain the competitive ratio of Balance on $G_5^\star$ and $G_6^\star$, which are $\frac{1}{5}f_3(1,1,3) \approx 0.604$ and $\frac{1}{6}f_3(1,1,4) \approx 0.599$, respectively. In addition, we can also obtain the competitive ratio of Balance on G_3 in (Mehta & Panigrahi, 2012), which is $\frac{1}{3}f_3(1,1,1) \approx 0.621$.

D. Agent `alg` Setup

D.1. MDP Formulation

Recall that the input instance is a probability matrix $\mathcal{P}$ sampled from π. In general, `alg` reads $\mathcal{P}$ column-by-column, and takes an action at state s_j to match v_j to which neighbor. We first highlight the main differences in OMSR to the RL agents for other OBM problems:

1. To achieve robustness, in the environment, all distributions learned by `adv` from previous iterations will be taken into consideration with a discounted factor, as well as a mixture of a random distribution.
2. In the construction of state space, besides the current graph patterns, i.e., the partial $\mathcal{P}$, we involve another observation: the number of failure attempts for each offline vertex, denoted by a vector L of size n. The main reason comes from the idea behind Balance: fewer failure attempts bring a larger marginal gain in success probability.
3. Because of the endogenous randomness from OMSR, i.e., a matching is successful with a certain probability p, special treatments are necessary: (1) an indicator on whether each offline vertex is successful must be observed in state space; (2) the transition is random. Different states will be observed according to whether the matching taken by an action is successful; (3) positive rewards are only given to successful matchings.

Environment. Suppose this is $(T+1)$-th iteration of training. Environment samples instances over a distribution π given by: with probability $\frac{\beta\gamma^{k-i}}{\sum_{j=0}^{k-1}\gamma^j}$ sampled from π_k; with probability $1-\beta$ sampled from a random distribution π_{rand}, where $\pi_k, k \in [T]$ represents the policy distribution learned by `adv` in k-th iteration, and β, γ are hyperparameters.

State Space. At timestep j, online vertex v_j arrives with the j-th column $\mathcal{P}_{[:,j]}$ of $\mathcal{P}$. A state s_j consists of $\mathcal{P}_{[:,j]}$ and two vectors of size n towards n offline vertices: L for the number of previous failure attempts and w for whether each offline vertex is successful or not. A terminal state $\hat{S}$ is reached when the complete $\mathcal{P}$ is revealed. The length of an episode is $T = m$.

Action space. An action $a_j \in A_j$ is a vector of size $n+1$, with only one component set as 1 and n components set 0, indicating that which neighbor is matched to v_j or leaving v_j unmatched.

Transition. s_j is transited to s_{j+1} with probability p if a_j produces a successful matching, and to s'_{j+1} with probability $1-p$ otherwise. s_{j+1} and s'_{j+1} differ in vectors L and w.

Reward. Reward r_{a_j} is set as 1 if a_j produces a successful matching, and 0 otherwise.

Policy. A stochastic policy $\pi(a_j|s_j)$ outputs a distribution over the actions in A_j, yielding an expected return $Q_\pi(s_j, a_j)$, which represents the expected future return obtained by taking action a_j in state s_j.

D.2. Training Algorithm

We use a simple DQN algorithm. Our model is a feed-forward neural network with two hidden layers and ReLU for non-linearity. The first layer contains $3n+2$ neurons and the second contains 2^{n+3} neurons. We set a fixed learning rate of 10^{-3} for the Adam optimizer and a batch size of $N_{\text{batch}} = 16$. For ϵ-greedy, we set $\epsilon = 0.2$.

D.3. Training for Unequal Probability Case

The training details for the algorithm agent in this case are consistent with the MDP formulation in Section 4. So we only discuss the differences in the adversary agent. Note that the equal probability case is a special case of the unequal probability case.

Recall that in OMSR, the input instance is a matrix $\mathcal{P}_{n\times m}$ for n offline vertices and m online vertices. The difference between the MDP in Section 3 and that in this case is that the values in the matrix $\mathcal{P}$ are not only p or 0, but are randomly selected. We typically randomly select a value k from the set $\{k_1, k_1+1, \cdots, k_2\}$, and set the probability of the edge to be $1/k$ or set the probability to be 0. k_1 and k_2 are hyperparameters. When $k_1 = k_2$, this special case corresponds to the equal probability case. The details of MDP formulation are presented as follows.

Environment. The current matching policy defined by `alg`, and a calculator on `CR`.

State Space. A state s_j at a timestep j is the current (partial) matrix of $\mathcal{P}$, i.e. $\mathcal{P}_{[:,1:j-1]}$. A terminal state $\hat{S}$ is reached when a complete $\mathcal{P}$ is generated. The length of an episode is $T = m$.

Action space. At state s_j, an action $a_j \in A_j$ taken by `adv` is to determine the probabilities in the j-th row. It selects a value in $\{0, 1/k_1, 1/(k_1+1), \cdots, 1/k_2\}$, and sets it as p_{ij}. The size of action space $|A_j|$ is $(k_2 - k_1 + 2)^n$.

Transition. The transition from state s_j to the next state s_{j+1} is *deterministic*. If action a_j is taken, the j-th column in matrix $\mathcal{P}$ is updated and moves on to the next episode step, $j+1$.

Reward. If an episode ends, all taken actions will receive a reward of $1 - \texttt{CR}$, where `CR` is the competitive ratio of the current algorithm environment running on the generated instance.

Policy. At state s_j, a stochastic policy $\pi(a_j|s_j)$ outputs a distribution on $(k_2 - k_1 + 2)^n$ cases.

E. Supplementary Experiments

We present more empirical evaluations of the trained hard instances and robust algorithms to demonstrate the effectiveness of our framework. Our codes are given in the supplementary material.

E.1. Experiments for Hard Instances Validation

The performance of algorithms on the worst-case instances in different iterations. In the equal probability case, we evaluate the performance of the learned algorithms in each iteration, against the corresponding outputs of the adversary agent. We compare these results to our baseline: Balance to show how Balance performs on these hard instances. In Figure 11, we plot the average competitive ratios as a function of the (equal) probability p on edges in each iteration. In general, the learned algorithms perform similarly to the Balance algorithm when $p < 0.5$. However, when $p > 0.5$, the learned algorithms significantly outperform Balance. As the number of iterations increased, the curves become smoother, and if fix p, the performance of algorithms get worse. This indicates that adv can generated hard instances in iterative training. Note that, as p tends to 1, Balance performs poorly under the outputs of adv with only 0.5-competitive. This also indicates that adv can learn worst-case instances for alg, since if $p = 1$, OBSR reduces to OBM, Balance is known to perform bad.

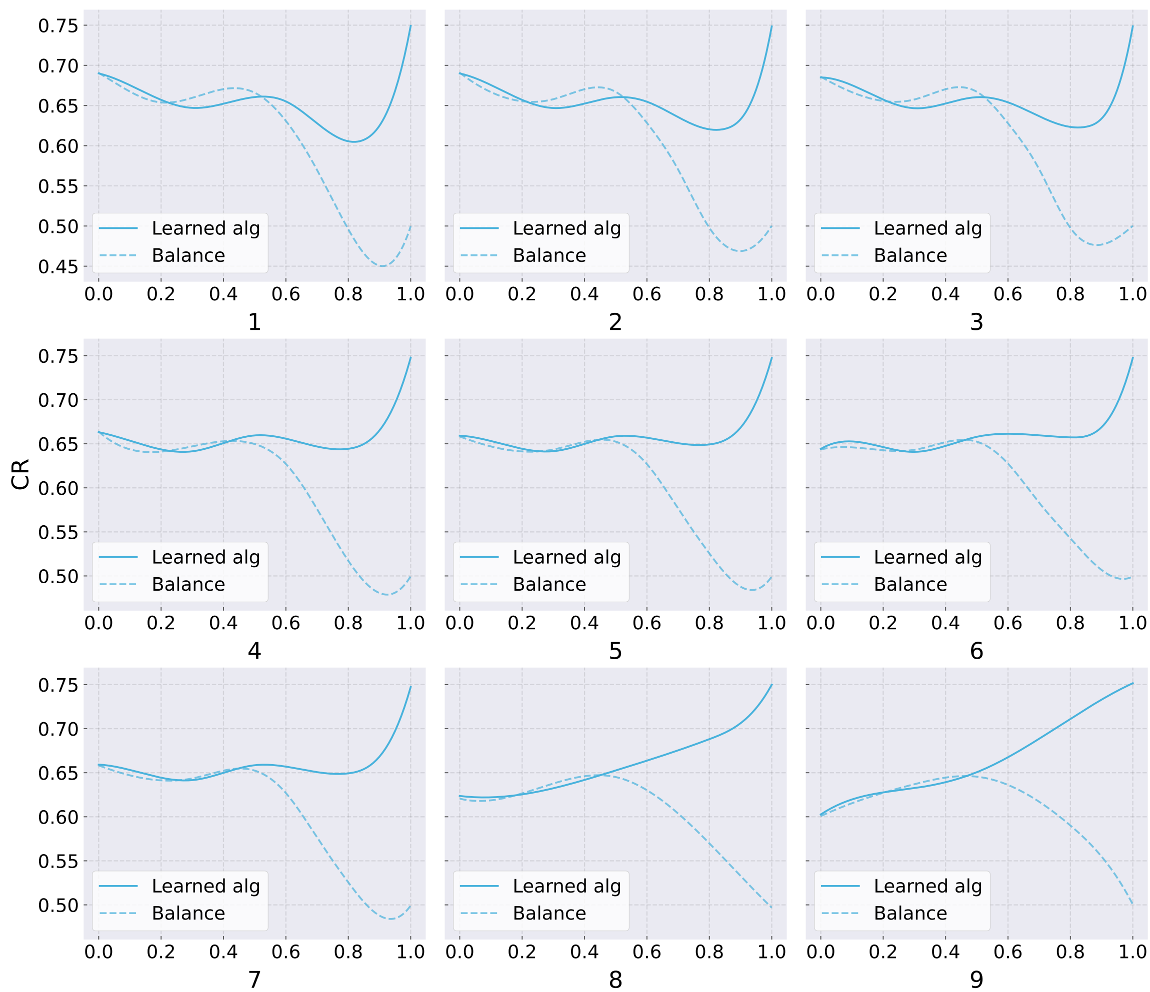


Figure 11. The average CRs of learned algorithms and Balance under the worst-case distributions in each iteration, with $m = n/p$ varying on p.

The performance of algorithms on known worst cases. To see the performances of Balance and our algorithms (from training iterations 1 to 10) on the worst instances given in both MP12 (Mehta & Panigrahi, 2012) (their G_3 and G_4) and this paper (G_6^* and G_7^* in Figure 10). Figure 12 plots the results. The algorithm learned by alg can converge to optimal CRs on these instances. It shows that the worst-case instances generated by adv make the algorithm perform worse than other known hard instances.

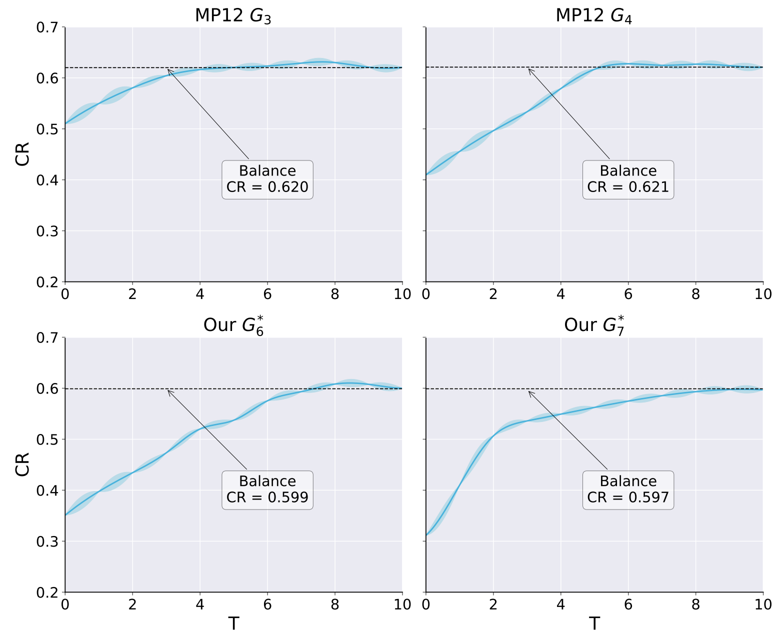


Figure 12. The CRs of learned algorithms from iteration 0 to 10 running on some specific worst-case instances. As a comparison, the dashed line represents the CR of Balance.

E.2. Experiments for Robust Algorithms Evaluation

The performance of algorithms on baselines. We take experiments and evaluations on some instances which are well known to be hard for OBM and related problems, including triangular graphs and thick-z graphs. We also test the average performance on some randomly generated instances. These instances are formally defined in Appendix B.1.

We test the learned algorithm in each iteration T on the triangular graphs. Figure 13 plots the average CRs as a function of iteration T, in both equal and unequal cases.

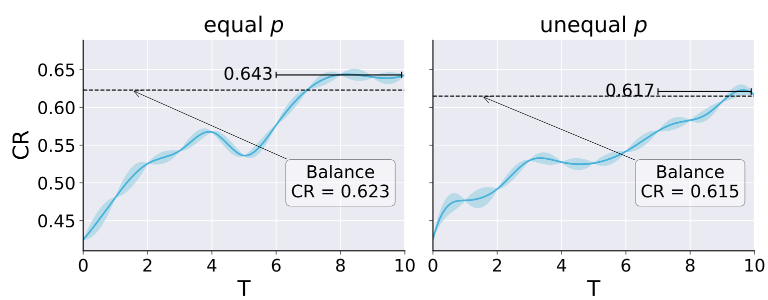


Figure 13. The average CRs of algorithms on triangular graphs in iteration 0 to 10. The curve plots the CR as a function of the iteration T. As a comparison, the dashed line represents the CR of Balance (0.623 for equal and 0.615 for unequal).

We test the learned algorithm in each iteration T on the thick-z graphs. Figure 14 plots the average CRs as a function of iteration T, in both equal and unequal cases.

We test the learned algorithm in each iteration T on the random graphs. Figure 15 plots the average CRs as a function of iteration T, in both equal and unequal cases.

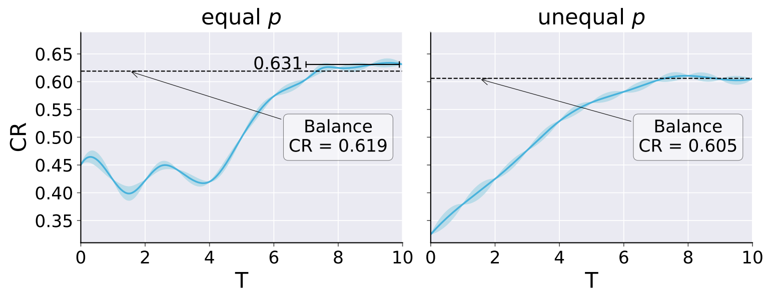


Figure 14. The average CRs of algorithms on thick-z graphs in iteration 0 to 10. The curve plots the CR as a function of the iteration T. As a comparison, the dashed line represents the CR of Balance (0.619 for equal and 0.605 for unequal).

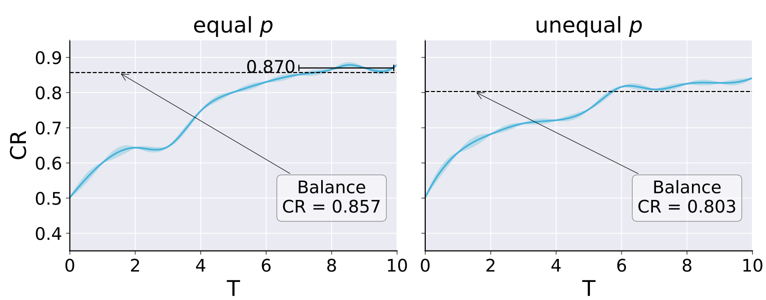


Figure 15. The average CRs of algorithms on random graphs in iteration 0 to 10. The curve plots the CR as a function of the iteration T. As a comparison, the dashed line represents the CR of Balance (0.857 for equal and 0.803 for unequal).