Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach

Yansheng Wang †, Yongxin Tong †, Cheng Long ‡, Pan Xu #
, Ke Xu †, Weifeng Lv †
†BDBC, SKLSDE Lab, School of Computer Science and Engineering and IRI, Beihang University, China
‡School of Computer Science and Engineering, Nanyang Technological University, Singapore
#Department of Computer Science, University of Maryland, College Park, USA
†{arthur_wang, yxtong, kexu, lwf}@buaa.edu.cn, ‡c.long@ntu.edu.sg, #panxu@cs.umd.edu

Abstract—**Online bipartite graph matching is attracting growing research attention due to the development of dynamic task assignment in sharing economy applications, where tasks need be assigned dynamically to workers. Past studies lack practicability in terms of both problem formulation and solution framework. On the one hand, some problem settings in prior online bipartite graph matching research are impractical for real-world applications. On the other hand, existing solutions to online bipartite graph matching are inefficient due to the unnecessary real-time decision making. In this paper, we propose the *dynamic bipartite graph matching (DBGM)* problem to be better aligned with real-world applications and devise a novel adaptive batch-based solution framework with a constant competitive ratio. As an effective and efficient implementation of the solution framework, we design a reinforcement learning based algorithm, called *Restricted Q-learning (RQL)*, which makes near-optimal decisions on batch splitting. Extensive experimental results on both real and synthetic datasets show that our methods outperform the state-of-the-arts in terms of both effectiveness and efficiency.**

I. Introduction

With the rapid development of sharing economy, many new business models are springing up and attracting much popularity. Typical real-world applications include intelligent transport platforms *e.g.*, Uber and crowdsourcing platforms *e.g.*, Amazon Mechanical Turks (AMT). One central issue in these emerging applications is *dynamic task assignment*, which aims to assign each worker with one or more tasks to maximize the overall revenue of the platform, where the workers are *dynamic* while the tasks arrive *sequentially*.

Dynamic task assignment can be formulated as an online bipartite matching problem, which has attracted growing research interest [1]–[4]. Earlier studies [1], [2] focus on one-sided online bipartite matching, where tasks are assumed to come dynamically while workers are considered static. A few recent efforts [3], [4] have explored the two-sided online bipartite matching problem, where both tasks and workers arrive dynamically. However, some of their problem settings are still restrictive and fail to model emerging sharing economy applications. For instance, in [3], the authors only maximize the total number of assignments, while real-world applications often focus on more general goals such as maximizing the overall profits. In [4], it is assumed that the deadline of each worker/task is known, while in applications such as on-demand taxi dispatching, workers and tasks usually would not report their deadlines to the platform. To be better aligned with these practical settings, we propose a more generic problem called the *dynamic bipartite graph matching (DBGM)* problem. In the DBGM problem, both workers and tasks arrive dynamically and can leave at any time without notifications in advance, and the goal is to find a matching allocation that yields the highest total revenue.

In addition to the restrictive problem settings, prior solutions to online bipartite graph matching [4]–[9] are also impractical to real-world sharing economy applications. They often rely on strong assumptions such as specific demand distributions or market equilibrium and some of them are low in efficiency, *e.g.*, of cubic time complexity [4]. The strong assumptions and the low efficiency are the results of real-time matching decisions upon arrival of a single task or worker, which may be unnecessary in practice. For example, passengers and taxi drivers are likely to wait for a short time before they disappear from the on-demand taxi-calling applications. This observation leads us to take an alternative framework to solve the DBGM problem, *i.e.*, in a *batch-based* manner.

More specifically, we propose an *adaptive batch-based framework* to the DBGM problem. The idea is to cumulate the dynamically coming tasks and workers into a batch, match those in the batch when the batch size is suitable and repeat the process. There are two critical challenges in designing an effective adaptive batch-based framework to the DBGM problem. *(i) How optimal is an adaptive batch-based framework in theory? (ii) How to implement an optimal strategy to split batches with performance guarantees?* For the first challenge, we conduct a theoretical analysis and prove that if the in-batch matching algorithm outputs a local optimum *e.g.*, via the Hungarian algorithm [10], then there exists an adaptive batch-based strategy that guarantees the overall performance. For the second challenge, we model the strategy searching problem as a sequential decision making problem [11], because the decision of whether to split the current batch is made at each time step sequentially. Particularly, we formulate the problem as a *Markov decision process (MDP)* [12] with unknown parameters, which can be solved by *reinforcement learning (RL)* [13]. Then we propose an effective and efficient RL-based algorithm to solve the MDP with unknown parameters in the context of our DBGM problem.

In summary, the main contributions of this work are:

- We propose the *dynamic bipartite graph matching (DBGM)* problem, which is a more practical formulation of dynamic task assignment in emerging intelligent transportation and spatial crowdsourcing applications.
- We devise a novel adaptive batch-based framework to solve the DBGM problem and prove that its performance is guaranteed by a constant competitive ratio $\frac{1}{C-1}$ under the adversarial model , where C is the maximum duration of a worker/task.
- We propose an *effective and efficient* RL-based algorithm, Restricted Q-learning (RQL), to retrieve a near-optimal batch-based strategy.
- We validate the effectiveness and efficiency of our methods on synthetic and real datasets. Experimental results show that our methods outperform state-of-the-arts in terms of the overall revenue and running time.

In the rest of this paper, we review related work in Sec. II and formally define the DBGM problem in Sec. III. We introduce the adaptive batch-based framework and analyze its competitive ratio in Sec. IV and propose an RL-based solution to find the batch splitting strategy in Sec. V. We evaluate our solutions in Sec. VI and finally conclude in Sec. VII.

II. Related Work

We study a generic *online maximum bipartite matching* problem and adopt a *reinforcement learning* based algorithm for our adaptive batch-based solution framework. Below we review representative work in these two categories of research.

A. Online Maximum Bipartite Matching

Online maximum bipartite matching problems include one-sided version and two-sided version, and both versions consider the weighted or the unweighted (*i.e.*, maximum cardinality matching) case. We divide the algorithms to these problems into two categories according to the analytical model they use, *i.e.*, the adversarial model and the stochastic model. Algorithms in the adversarial model always make guarantees on the worst case performance while those in the stochastic model need more assumptions and care about expected performance. As a result, conclusion made in the adversarial model still holds in the stochastic model.

1) Algorithms in the Adversarial Model: In this model, the information on the graph and the arrival order of nodes are unknown and can be arbitrarily bad. In [1], the authors study the one-sided online maximum unweighted bipartite matching and propose the randomized algorithm called Ranking with a competitive ratio of $1 - \frac{1}{e}$ in adversarial model. Another randomized algorithm named Greedy-RT for the one-sided weighted bipartite matching is proposed in [5]. The two-sided unweighted case is considered in [6], where a 0.526-competitive algorithm is presented. A recent work [3] extends the Ranking algorithm [1] to the two-sided unweighted case and shows a competitive ratio of 0.5211. To the best of our knowledge, no prior work has studied the general two-sided weighted case in the adversarial model.

2) Algorithms in the Stochastic Model: This model includes the random order model and the I.I.D model. The former assumes that the arrival order of nodes follows a uniform random permutation, and cares about the expected performance of online algorithms [14] while the latter assumes that there is an underlying distribution of the coming objects [15]–[17]. The Ranking algorithm is 0.6534-competitive in the one-sided weighted case in the random order model [18]. In [7], the authors study the two-sided maximum cardinality matching on spatial data and a 0.47-competitive method under the I.I.D model is designed based on predictions of tasks and workers. In [4], the authors propose the TGOA algorithm with a $\frac{1}{4}$ competitive ratio in the random order model. Our work is closely related to [4] in that we both study the two-sided online maximum weighted bipartite matching. Our work differs from [4] in that *(i)* we do not assume the deadline of each worker/task is known and *(ii)* we propose solutions in the adversarial model.

B. Reinforcement Learning

Reinforcement learning (RL) [13] studies how agents should take actions in an unknown environment to maximize a cumulative reward. The environment can be formulated as a Markov Decision Process (MDP) [12]. There are mainly three categories of RL algorithms. Algorithms that aim to learn the parameters of an unknown MDP are called model-based algorithms [19], [20], while those make no efforts to learn a model are called model-free, and the third type is policy search such as policy gradient [21]. We adopt model-free algorithms since they are more computationally efficient and have guarantees on convergence. Particularly, we utilize Q-learning [22], which estimates the Q-function iteratively using Bellman backups [23] and acts greedily according to the Q-function, and can converge to optimum.

Some pioneer studies have applied RL to matching problems, such as advertisement placement [24] and spatial crowdsourcing [25], [26]. In [24], the authors propose the combinational multi-armed bandit (CMAB) framework, which decides whether to match the coming nodes in the one-sided online maximum bipartite matching problem. The algorithms in [25] focuses on learning parameters like worker reliability using methods from the CMAB class in spatial crowdsoucring tasks, while the structure of bipartite graphs is still static. In [26], the authors also model the online matching problem as an MDP but with a straightforward idea. It takes every matching behavior as an action, which makes the approach impractical due to the high sparsity problem.

In this paper, we design effective RL-based algorithm to implement our adaptive batch-based solution framework to the DBGM problem.

III. Problem Statement

This section defines the DBGM problem (Sec. III-A) and important concepts such as the competitive ratio in the adversarial model (Sec. III-B). Finally we present a Greedy baseline algorithm to the DBGM problem (Sec. III-C). Table I lists

TABLE I: Summary of symbol notations

Notation	Description
B	The dynamic bipartite graph
L, R	The set of left and right nodes on the graph
E	The set of edges on the graph
$i.d$	The duration of a node i
$w(i,j)$	The weight of edge between node i and j
$\mathcal{M}$	The matching allocation set over the graph
$U(B,\mathcal{M})$	The utility score of $\mathcal{M}$ over B
$Opt(B)$	The offline optimum of B
S_B	The adaptive batch-based strategy space
$\boldsymbol{\sigma}$	The adaptive batch-based strategy
$b(i,j)$	The function to indicate whether $(i,j]$ is a batch
$U_R(B,\boldsymbol{\sigma})$, $U_E(B,\boldsymbol{\sigma})$	The utility score induced by a strategy in remain/expiration model
C	The upper bound of the duration
S_B^{Len}	The restricted strategy space of S_B by an interval Len
l_{min}, l_{max}	The minimum/maximum length of batch

the notations used throughout the paper. Some notations are inspired by [27], [28].

A. Problem Definition

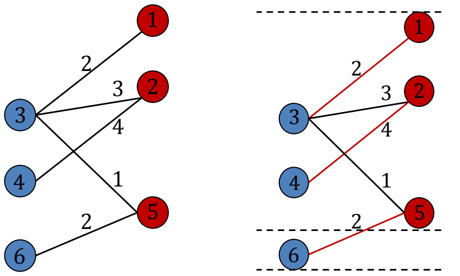
Definition 1 (Dynamic Bipartite Graph, DBG). *A dynamic bipartite graph is defined as $B = (L, R, E)$, where $L = \{i \in \mathbb{N}^*\}$ and $R = \{j \in \mathbb{N}^*\}$ are the sets of left and right nodes with $L \cap R = \emptyset$ and $E \subseteq L \times R$ is the set of edges between L and R. Each node $i \in L$ ($j \in R$) has its arriving time at i (j). The notations are abused to denote both the nodes and the nodes' arriving times for easy reference. Each node i (j) also has a duration denoted by $i.d$ ($j.d$), meaning the amount of time it keeps activated. Each edge $(i,j) \in E$ has a weight denoted by e_{ij}. For convenience, B also denotes the set of all its nodes, and we have $|B| = |L| + |R|$.*

We do not specify the unit of arriving time and duration, as it is application-specific. For instance, the unit can be one second in on-demand taxi dispatching. We also assume that the arriving time of each node is different for the simplification of analysis. Our solution works with or without this assumption.

Note that there is no need to consider those edges (i,j) with $[i, i+i.d] \cap [j, j+j.d] = \emptyset$ for matching since in this case one node expires before the other arrives, and thus we focus on the remaining edges only. Specifically, we define a weight function $w : L \times R \to \mathbb{R}$ such that $w(i,j) = e_{ij}$, if the edge (i,j) exists in E and $[i, i+i.d] \cap [j, j+j.d] \neq \emptyset$, and $w(i,j) = 0$ otherwise. In the former case, we have $w(i,j) > 0$. With this, in the following, we could safely focus on those edges (i,j) with $w(i,j) > 0$ for matching.

Definition 2 (Matching Allocation). *A matching allocation over a dynamic bipartite graph B is denoted by $\mathcal{M} = \{(i,j) | i \in L, j \in R\}$. It is a set of node pairs where each node appears at most once.*

Definition 3 (Utility Score). *The utility score of a matching allocation $\mathcal{M}$ over a dynamic bipartite graph B is measured by $U(B, \mathcal{M}) = \sum_{(i,j) \in \mathcal{M}} w(i,j)$.*



(a) An example of DBG

(b) A batch-based strategy

Fig. 1: A simple example of DBG and batch-based strategy

Example 1. *Take Fig. 1a as an example. The indices of the 6 nodes are their arriving times,* i.e.*, $1, 2, 3, 4, 5, 6$ and their durations are $3, 5, 3, 1, 2, 4$ respectively. Suppose there are edges between node 1 and 4, also 4 and 5. Note that the duration of node 1 is 3, which means it will vanish before node 4 appears. Thus we have $w(1,4) = 0$. The same situations happen for $w(4,5)$ and $w(2,6)$. The other edge weights are shown in the figure, and we have $w(1,3) = 2$, $w(2,4) = 4$, etc.. A possible matching allocation in Fig. 1a is $\mathcal{M}_1 = \{(2,4), (3,5)\}$. We have $U(B, \mathcal{M}_1) = 4 + 1 = 5$.*

Definition 4 (DBGM Problem). *Given a dynamic bipartite graph B, the DBGM problem is to find a matching allocation $\mathcal{M}$ to maximize the utility score,* i.e.*, $\max_{\mathcal{M}} U(B, \mathcal{M})$ in the online scenario.*

Here in the online scenario, nodes arrive one by one following the arriving times in B, and will vanish after their deadlines. The arriving time and vanishing time of any node cannot be forecast. The difference between DBGM and the two-sided online maximum bipartite matching [4] is that the nodes in DBGM are totally dynamic, but in the latter, the deadline of each node is assumed given upon its arrival. Hence the DBGM problem is more generic and better aligned with real-world applications.

An online algorithm to the DBGM problem decides at each time step k a subset of $\mathcal{M}$, denoted by $\mathcal{M}_k$, which only contains the activated nodes that are not expired yet at k. The final matching allocation is $\mathcal{M} = \bigcup_{k=1}^{T} \mathcal{M}_k$, where T is the number of time steps.

B. Evaluation Metric

The performance of an online algorithm is often accessed by comparing against the optimal results in the offline scenario. In the following, we will introduce the offline optimum of the DBGM problem and the competitive ratio in the adversarial model. Note that the competitive ratio in the adversarial model will still be guaranteed in the stochastic model.

A dynamic bipartite graph B can be adjusted to a general bipartite graph B' by removing all the edges that do not satisfy the restriction of deadline, $[i, i + i.d] \cap [j, j + j.d] \neq \emptyset$ (*i.e.*, only those edges with $w(i,j) > 0$ are kept). Then the offline optimum is defined by $Opt(B) = \max_{\mathcal{M}} U(B', \mathcal{M})$.

Algorithm 1: Greedy Algorithm

```
input : A dynamic bipartite graph $B$
output: A matching allocation $\mathcal{M}$
$\mathcal{M} \leftarrow \emptyset$;
for each new arrival node $u$ in $B$ do
    if $u \in L$ then
        $V \leftarrow \{v | v \in R, w(u, v) > 0\}$;
        if $V \neq \emptyset$ then
            $v' \leftarrow \arg\max_{v \in V} w(u, v)$;
            Remove $u$, $v'$ from $B$;
            $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, v')\}$;
    else
        Symmetrically matching $u \in R$;
return $\mathcal{M}$
```

The optimal matching allocation in the offline scenario can be found in polynomial time by Hungarian algorithm [10]. For example, in Fig. 1a, the optimal matching allocation is $\{(1, 3), (2, 4), (5, 6)\}$ and $Opt(B) = 8$.

Competitive Ratio. The competitive ratio in the adversarial model of an online algorithm for the DBGM problem is the minimum ratio between the utility score of matching allocation $\mathcal{M}$ produced by the algorithm and the offline optimum over all possible instances of the dynamic bipartite graph,

$$CR = \min_{B} \frac{U(B, \mathcal{M})}{Opt(B)}$$

The competitive ratio in the adversarial model measures the performance of an online algorithm in the worst case, and we are interested in online algorithms which have guarantees on the lower bound of the ratio.

C. Greedy Algorithm: A Baseline

A straightforward solution to the DBGM problem is the Greedy algorithm, which matches a node to its neighbor with the maximum edge weight, and leaves a node waiting if it does not have any available neighbors yet. The procedure is illustrated in Algorithm 1. By conducting the Greedy Algorithm in Fig. 1a, we will get a matching allocation $\{(2, 3), (5, 6)\}$ and the utility score is 5.

The competitive ratio of the Greedy algorithm can be arbitrarily bad in the adversarial model, because we can insert a node v into the opposite side of a node u with a potential weight of 1 as soon as u is just matched greedily to some other node with the weight ϵ. If we set ϵ to an infinitely small value, then the competitive ratio is close to 0.

IV. Solution Framework

This section presents the adaptive batch-based framework (Sec. IV-A) and analyzes its achievable optimum (Sec. IV-B).

A. Adaptive Batch-based Framework

The rationale of our adaptive batch-based framework to the DBGM problem lies in two-fold. *(i)* If nodes were allowed to wait rather than being immediately assigned, they may meet better matching candidates in the future (*i.e.*, the advantage of batch-based decisions). *(ii)* The batch size should be adjusted according to the current dynamic bipartite graph (*i.e.*, the need for an adaptive batch size). In this work, we propose to decide the batch size on the fly by using a binary decision indicator for each time step. Specifically, if the indicator is 0, all the nodes will wait at this time step; otherwise we will match all the activated nodes in the waiting pool. We formally define the adaptive batch-based strategy below.

Definition 5 (Adaptive Batch-based Strategy)**.** *Denote the strategy space of B as $S_B = 2^{|B|+1}$. Each adaptive batch-based strategy $\boldsymbol{\sigma} = (\sigma_k)_{k \in [\![0, |B|]\!]}$ is a binary sequence with length $|B| + 1$. Let $\sigma_0 = 1$. $\forall 1 \leq k \leq |B|$, if $\sigma_k = 1$, the strategy is to match all the nodes available (including itself) to achieve a local optimum (*e.g.*, by Hungarian algorithm). Otherwise, all nodes are kept till the next time step. An interval $(i - 1, j]$ is called a batch if $\sigma_{i-1} = \sigma_j = 1$, and $\forall k \in [\![i, j - 1]\!], \sigma_k = 0$. We set a binary function $b(i, j) = 1$ if $(i - 1, j]$ is a batch; otherwise, $b(i, j) = 0$.*

We make the following notes on the definition above.

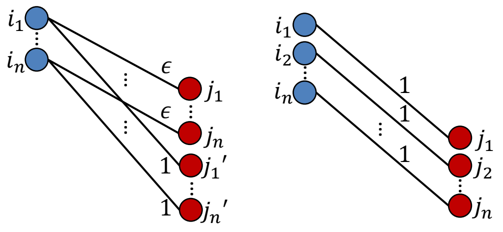
- If each σ_k is set to 1, it is equivalent to the Greedy algorithm in Sec. III-C. Hence the optimal adaptive batch-based strategy in the strategy space is always no worse than the Greedy algorithm.
- We perform optimal matching in each batch to achieve theoretical guarantees (see Sec. IV-B).
- The adaptive batch-based strategy consists of two models based on whether the unmatched nodes in each batch are removed (denoted as *expiration model*) or kept to the next batch (denoted as *remain model*). We analyze these two models separately.

We still use utility score to assess the performance of a strategy. We override the utility score function $U(B, \mathcal{M})$ by $U(B, \boldsymbol{\sigma})$ because a fixed strategy $\boldsymbol{\sigma}$ results in a deterministic matching allocation.

Definition 6 (Utility Score of Strategy)**.** *Let $U_R(B, \boldsymbol{\sigma})$ denote the utility score in the remain model,* i.e.*, the sum of weights in B induced by $\boldsymbol{\sigma}$ in the online scenario where the unmatched nodes in each batch will remain. Similarly, denote $U_E(B, \boldsymbol{\sigma})$ as the utility score in the expiration model,* i.e.*, the sum where the unmatched nodes in each batch will expire.*

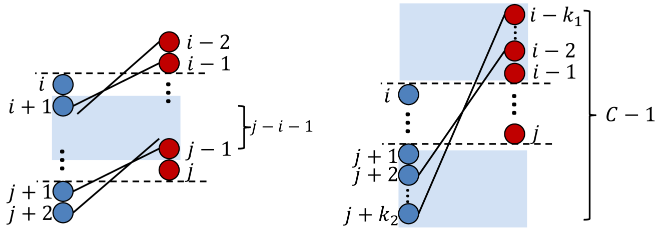
We further define the utility score of a strategy space in the remain and the expiration models: $U_R(B, S_B) = \max_{\boldsymbol{\sigma} \in S_B} U_R(B, \boldsymbol{\sigma})$ and $U_E(B, S_B) = \max_{\boldsymbol{\sigma} \in S_B} U_E(B, \boldsymbol{\sigma})$. The utility score of a strategy space represents the highest utility a strategy in that space can achieve.

Example 2. *Take Fig. 1b as an example. Suppose $\boldsymbol{\sigma}_1 = (1, 0, 0, 0, 0, 1, 1)$, there are two batches $(0, 5]$ and $(5, 6]$ as shown in the figure. If we remove the unmatched nodes*



(a) A counterexample (b) The example of tightness

Fig. 2: Two special graphs



(a) $A_1(i,j)$ (b) $A_2(i,j)$

Fig. 3: Illustrations on A_1 and A_2

in each batch, the matching allocation is $\{(a,c),(b,d)\}$ *and* $U_E(B,\boldsymbol{\sigma}_1)=6$. *Otherwise we have the allocation* $\{(a,c),(b,d),(e,f)\}$ *with* $U_R(B,\boldsymbol{\sigma}_1)=8$, *which is the optimal allocation. Different strategies have different utility scores. In this example,* $U_R(B,S_B)=U_R(B,\boldsymbol{\sigma}_1)=8$. *Yet* $U_E(B,S_B)\neq U_E(B,\boldsymbol{\sigma}_1)$, *because if we set* $\boldsymbol{\sigma}_2=(1,0,0,0,1,0,1)$, *we will have* $U_E(B,\boldsymbol{\sigma}_2)=8>U_E(B,\boldsymbol{\sigma}_1)$.

Next we study the theoretically achievable optimum of our adaptive batch-based framework by analyzing the competitive ratio (with respect to utility score) of the strategy space.

B. Theoretical Analysis on Achievable Guarantees

1) Assumptions: We assume that the duration of each node has an upper bound $C\in\mathbb{N}^*$ and $C\geq 2$. On the one hand, an upper bounded C is reasonable since no worker or task would wait endlessly in real applications. On the other hand, if $C=1$, no match exists as we assume the expiration comes before the arrival of new nodes at the same step. We demonstrate the necessity of this assumption via the following example.

Example 3. *Suppose the nodes in Fig. 2a come in the order of* $i_1,\cdots,i_n$, $j_1,\cdots,j_n$, $j_1',\cdots,j_n'$. *The weights of edges between* i_k *and* j_k *are* ϵ, *and those between* i_k *and* j_k' *are* 1. *The durations of* i_k *and* j_k *are infinite and the duration of* j_k' *is* 1. *No matter how we segment the batches,* $\frac{U_E(B,S_B)}{Opt(B)}=\frac{U_R(B,S_B)}{Opt(B)}=\frac{1+(n-1)\epsilon}{n}$, *which can be arbitrarily bad if* ϵ *approaches* 0. *Hence we assume that no duration can be larger than* C.

2) Main Results: We have the following two theorems on the competitive ratios of the strategy spaces in the remain and the expiration models.

Theorem 1. *For any given dynamic bipartite graph* B *with an upper bound of duration* $C\geq 2$, *we have*

$$\min_B \frac{U_E(B,S_B)}{Opt(B)}=\frac{1}{C-1} \qquad (1)$$

Proof. We first prove that $\min_B\frac{U_E(B,S_B)}{Opt(B)}\geq\frac{1}{C-1}$. The idea is to construct a good enough batch-based strategy.

Suppose $\mathcal{M}_B$ is the set of edges that are in the offline optimal matching allocation of B. Let e_{pq} be an edge in $\mathcal{M}_B$ where p comes earlier than q. We also use e_{pq} to represent the weight on the edge.

We first consider the special case where $C=2$. In this case, each node must be matched no latter than one round after its arrival. Thus our strategy is to set every $\sigma_q=1$ for each $e_{pq}\in\mathcal{M}_B$, and the strategy always leads to an optimal matching allocation. So $\frac{U_E(B,S_B)}{Opt(B)}=1=\frac{1}{C-1}$ if $C=2$.

Next we consider the general cases. For any batch $(i-1,j]$, we define two sets, $A_1(i,j)$ and $A_2(i,j)$ on it: $A_1(i,j)=\{e_{pq}\in\mathcal{M}_B|(i<p<j\wedge q>j)\vee(p<i\wedge i<q<j)\}$ and $A_2(i,j)=\{e_{pq}\in\mathcal{M}_B|p<i\wedge q>j\}$.

Intuitively, $A_1(i,j)$ contains edges with one end point in the batch and the other out of the batch while edges in $A_2(i,j)$ have two end points both out of the batch but in the different side. Thus if $C=3$, we have $|A_1(i,j)|\leq 1$ (as the maximum time span of an edge is 2) and $|A_2(i,j)|=0$. In the general case $C>3$, we take Fig. 3 for illustration. In Fig. 3a, all the edges in A_1 must have exactly one end point in the shaded area. As there are at most $(j-i-1)$ nodes in the shaded area, we have $|A_1(i,j)|\leq j-i-1$. In Fig. 3b, the same condition holds if $k_1=k_2$. The shaded area has a maximum span of $C-1$. Thus there are at most $C-1-(j-i+1)$ nodes in the shaded area and $|A_2(i,j)|\leq C-1-(j-i+1)$. In summary, $|A_1(i,j)|+|A_2(i,j)|\leq C-2$.

Now a good enough strategy $\boldsymbol{\sigma}^*$ is as follows:

- Initially, set $\sigma_k=0$ for $k\in[\![1,|B|]\!]$.
- Pick the edge with the largest weight in $\mathcal{M}_B$, named e_{ij}, and remove it from $\mathcal{M}_B$.
- Set $\sigma_{i-1}=\sigma_j=1$, and then remove all edges in $A_1(i,j)$ and $A_2(i,j)$ from $\mathcal{M}_B$.
- Repeat $(2),(3)$ until there is no edge in $\mathcal{M}_B$.

Then we prove that the competitive ratio of $\boldsymbol{\sigma}^*$ has a lower bound. Suppose in each batch $(i-1,j]$, by conducting the Hungarian algorithm, we have a gain of weights denoted by $u(i,j)$. Apparently, $u(i,j)\geq e_{ij}\geq e_{pq}$ where $e_{pq}\in A_1(i,j)\cup A_2(i,j)$. Consider the edges we drop in our strategy. The sum of weights dropped by edges in $A_1(i,j)$ and $A_2(i,j)$ are denoted by $u_1(i,j)$ and $u_2(i,j)$, respectively. We have $u_1(i,j)\leq|A_1(i,j)|e_{ij}$, $u_2(i,j)\leq|A_2(i,j)|e_{ij}$. Thus, $\forall B$, we have

$$\begin{aligned}
Opt(B) = &\sum_{e_{pq}\in\mathcal{M}_B} e_{pq} = \sum_{b(i,j)=1} (u_1(i,j)+u_2(i,j)+e_{ij})\\
\leq &\sum_{b(i,j)=1} (|A_1(i,j)|+|A_2(i,j)|+1)e_{ij}\\
\leq &\sum_{b(i,j)=1} (C-2+1)e_{ij}\\
\leq &(C-1)\sum_{b(i,j)=1} u(i,j)\\
\leq &(C-1)U_E(B,\boldsymbol{\sigma}^*)\\
\leq &(C-1)U_E(B,S_B)
\end{aligned}$$

In summary, $\min_B \frac{U_E(B,S_B)}{Opt(B)} \geq \frac{1}{C-1}$. Notice that the same inequality also holds for $U_R(B,S_B)$, because we have $\sum_{b(i,j)=1} u(i,j) = U_E(B,\boldsymbol{\sigma}^*) \leq U_R(B,\boldsymbol{\sigma}^*)$ by the same construction of $\boldsymbol{\sigma}^*$.

To prove the bound is tight, we use the example in Fig. 2b. In the graph, each node has the duration equal to $n+1$, and as a result, $C=n+1$. In this case, $Opt(B)=C-1$. No matter how we change the strategy, $U_E(B,S_B)=1$. This verifies that the bound $\frac{1}{C-1}$ is tight. □

Theorem 2. *For any given dynamic bipartite graph B with an upper bound of duration $C \geq 3$, we have*

$$\frac{1}{C-1} \leq \min_B \frac{U_R(B,S_B)}{Opt(B)} < \frac{2}{C-2} \tag{2}$$

Proof. If $C=2$, $U_R(B,S_B)$ is also optimal. We only consider the general cases when $C\geq 3$ here. As demonstrated above, we have $\min_B \frac{U_R(B,S_B)}{Opt(B)} \geq \frac{1}{C-1}$. To find an upper bound, we take Fig. 2a as an example. We set $n=\lfloor\frac{C-1}{2}\rfloor$. As pointed out above, $\frac{U_R(B,S_B)}{Opt(B)} = \frac{1+(n-1)\epsilon}{n} < \frac{1}{n}+\epsilon \leq \frac{2}{C-2}+\epsilon$. Since ϵ can be infinitely small, thus $\min_B \frac{U_R(B,S_B)}{Opt(B)} < \frac{2}{C-2}$. □

Summary. The best strategy in the expiration model can achieve a constant competitive ratio, and that of the best strategy in the remain model also lies in a constant interval. Hence the adaptive batch-based framework has strong performance guarantees in theory. Next we design an effective and efficient strategy from the strategy space by exploiting reinforcement learning techniques.

V. RL-based Solution

Sec. IV shows that the adaptive batched-based framework has theoretical performance guarantees to the DBGM problem. This section aims to devise an algorithm following such a framework. We first explain the high-level idea (Sec. V-A), then model the algorithm as a Markov decision process (Sec. V-B), and finally present the a reinforcement learning based algorithm implementation (Sec. V-C).

A. Basic Idea

As shown in Sec. IV-B, the performance guarantees of the adaptive batch-based framework require *(i)* an in-batch algorithm that outputs the local optimum and *(ii)* an optimal strategy to split batches. The former can be achieved by classical algorithms such as the Hungarian algorithm [10], while the latter is our focus.

We model the batch splitting process as a *Markov decision process (MDP)* [12] with unknown parameters, because it is a sequential decision making problem, and the actions we take interact with the environment (*i.e.*, the dynamic bipartite graph) and influence the reward (*i.e.*, the utility score).

To find an optimal strategy to split batches, we apply *Reinforcement Learning (RL)* [13], because it proves to be effective for the sequential decision making problem in MDPs with unknown parameters. Particularly, we propose the Restricted Q-learning (RQL) algorithm, which is based on Q-learning [22] but is optimized to have a notably smaller search space.

We present the MDP modeling and the details of our RQL algorithm in sequel.

B. MDP Modeling for Batch Splitting

We model the environment and actions with an MDP in the context of batch splitting as below.

Given a dynamic bipartite graph B, we define a finite-horizon undiscounted Markov decision process $M = (\mathcal{S},\mathcal{A},T_r,R_w,H)$ on B where

- $\mathcal{S}$ is the state space, where every state $s\in\mathcal{S}$ stands for a bipartite graph.
- $\mathcal{A}=\{0,1\}$ is the action space, where action 0 means to make no matches and go to the next time horizon and action 1 means to perform a maximum weight matching *e.g.*, using the Hungarian algorithm, drop the edges that have been matched, and go to the next time horizon.
- $T_r : (\mathcal{S}\times\mathcal{A}\times\mathcal{S}) \rightarrow [0,1]$ is the transition distribution, which models the dynamics of nodes joining in or leaving the graph. The probability of reaching s' from s by executing a is then expressed by $T_r(s,a,s')$.
- $R_w : (\mathcal{S}\times\mathcal{A}) \rightarrow \mathbb{R}^+$ is the reward function. If we conduct action a at state s, we will get a reward of $R_w(s,a)$, which is the sum of weights of matched pairs. The reward is a constant given s and a, because if $a=1$, $R_w(s,a)$ is the output from the Hungarian algorithm conducted at s, and if $a=0$, $R_w(s,a)=0$.
- H is the time horizon, and $H=(1,2,\cdots,|H|)$.

C. Restricted Q-learning (RQL) for Optimal Batch Splitting

Reinforcement learning (RL) is effective to find an optimal policy in unknown MDPs, where a deterministic stationary Markovian policy π is defined by a mapping from $\mathcal{S}$ to $\mathcal{A}$. Here policies can be measured by the state-value function (Q-function) $Q^\pi_M(s_t,a_t) = \mathbb{E}[\sum_{t'=t}^{|H|} r_{t'} | s=s_t, a=a_t]$, where s_t, a_t and r_t are the state, action and reward at time step t respectively. An optimal strategy search problem (optimal batch splitting in our case) can be solved by RL because the execution result of a policy on the MDP is the same as a strategy and maximizing the utility is equivalent to maximizing the cumulative rewards at the initial state.

We build our solution upon Q-learning [22], a classical model-free RL method with relatively low computational complexity. Q-learning approximates the Q-function directly based on the recursion in Eq.(3), where s_t is the current state, a_t is the action chosen based on the current Q-function, r_{t+1} is the obtained reward after taking action a_t and s_{t+1} is the next state. It has been demonstrated that a policy conducted greedily according to the optimal Q-function is also optimal. The convergence of Q-learning has been proven in [22].

$$\begin{aligned} &Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \\ &\alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \end{aligned} \tag{3}$$

We propose the Restricted Q-learning (RQL) algorithm to further improve the efficiency to solve our DBGM problem. Specifically, we adapt the naive Q-learning to our problem and significantly reduce its search space using the restriction principle based on our analysis in Sec. IV. Next we explain the three parts of our RQL algorithm: *state representation*, *restriction principle* and *state & action reformulation*.

- **State Representation.** We represent each state by the pair containing the number of left nodes and also that of right nodes, *i.e.*, $(|L|, |R|)$, rather than using a state-action table to record the Q-function as in traditional Q-learning algorithms. This is because the state-action table has two problems in our case: *(i)* The size of state space of our MDP is infinite, meaning we cannot record every state-action pair. *(ii)* The table suffers from the sparsity problem because it is almost impossible to encounter the same state twice. The effectiveness of our representation lies in the fact for the action $a = 1$, the reward, which is the sum of the weights of the edges that are matched, is closely correlated with the number of nodes on each side. As there are at most C nodes waiting either on the left or on the right, the size of state space is $C \times C$, which is much smaller.
- **Restriction Principle.** An important improvement of our RQL algorithm over traditional Q-learning is to exploit the restriction principle to reduce the size of the strategy space (which is $2^{|H|}$ in traditional Q-learning). The restriction principle is based on the restricted strategy space defined as follows.

  Definition 7 (Restricted Strategy Space)**.** *The restricted strategy space of S_B by an interval Len is denoted by S_B^{Len}. The strategies in S_B^{Len} satisfy that $\forall b(i,j) = 1, j - i \in Len$.*

  To maintain the performance guarantees of strategies in the restricted space, we need to consider the construction procedure of $\boldsymbol{\sigma}^*$ in Theorem 1. Suppose the batches in $\boldsymbol{\sigma}^*$ have a maximum length of l_{max} and a minimum length of l_{min}. Let $Len = [l_{min}, l_{max}]$. If we search for strategies in the restricted strategy space S_B^{Len}, it is guaranteed that the best strategy we can find has a constant competitive ratio $\frac{1}{C-1}$ because $\boldsymbol{\sigma}^* \in S_B^{Len}$. However, we do not know l_{min} and l_{max} in advance. A naive approach is to use the super set of Len, *i.e.*, $[1, C]$ as $l_{max} \le C$. Another method is to estimate them by conducting the construction method on historical data, which can be time-consuming. A better method is to consider them as hyper-parameters of the learning algorithm and the values are empirically tuned.
- **State & Action Reformulation.** The aim of state & action reformulation is to meet the constraint without violating the principle of Q-learning when applying the restriction principle.

  The original action space is $\{0, 1\}$. To meet the interval constraint of the restriction principle, we have to force the action to be 0 or 1 if the batch length is out of Len. This violates the principle of Q-learning, where the action should only be decided by the Q values. Accordingly, we cannot ensure the number of our mandatory assignments thus the convergence will stand no more.

  To overcome this problem, we reformulate states and actions and no mandatory assignments would happen. First, when the batch length is smaller than l_{min}, we take the state when the batch length is exactly l_{min} as the next state and skip the states in-between. In other words, our opportunity of decision only occurs when the batch length is no smaller than l_{min}. Then we use (s, l) instead of s to represent a state where $l \in [l_{min}, l_{max}]$ is the length of the batch so far, and the action $a \in \{0, 1\}$ is replaced by $l' \in [l_{min}, l_{max}]$, meaning the expected length of the current batch. Thus $Q((s, l), l')$ stands for the Q-value of cutting the batch with a length l' at state s while the current length is l. We initially set the Q-values with all $l > l'$ to be zero and others to be non-zero. Thus impossible actions will not happen.

  Example 4. *Suppose $l_{min} = 10$, $l_{max} = 60$, and the greedy action $\arg\max_{l'} Q((s_0, 15), l') = 20$. It means that at state s_0 with a batch length $l = 15$, the best action is to cut the batch at the 20^{th} step, so it is better to wait at the current step. The best action 13 is impossible due to the initial rules. Also, if $l = 60$, the greedy choice of l' can only be 60. So the batch will be cut at its maximum size $l_{max} = 60$. The interval constraint is still satisfied.*

  According to the reformulations, the updates of Q-values for $l_t = l'_t$ and $l_t \neq l'_t$ are formulated as:

$$\begin{aligned} &Q((s_t, l_t), l'_t = l_t) \leftarrow Q((s_t, l_t), l'_t) + \alpha[r_t + \\ &\max_{l_0} Q((s_{t+l_{min}}, l_{min}), l_0) - Q((s_t, l_t), l'_t)] \end{aligned} \tag{4}$$

$$\begin{aligned} &Q((s_t, l_t), l'_t \neq l_t) \leftarrow Q((s_t, l_t), l'_t) + \\ &\alpha[\max_{l_0} Q((s_{t+1}, l_t + 1), l_0) - Q((s_t, l_t), l'_t)] \end{aligned} \tag{5}$$

Algorithm Sketch. Algorithm 2 illustrates our RQL algorithm. We initialize the Q-function with small positive real numbers (line 1) and set the Q-values of the situations where $l > l'$ to be 0 (line 2). In the outer loop of iterating the episodes, we first conduct action $a = 0$ for $l_{min} - 1$ times,

Algorithm 2: Restricted Q-learning

```
   input  : learning rate $\alpha \in (0, 1]$, $l_{min}$, $l_{max}$, $C$
   output: Learned state-value function $Q((s, l), l')$
1  $Q((s, l), l') \leftarrow Random(), \forall s, l, l'$;
2  $Q((s, l), l') \leftarrow 0, \forall s, \forall l > l'$;
3  for episode $1, 2, \cdots$ do
4      Repeatedly take action $a = 0$ for $l_{min} - 1$ times, observe $s_{l_{min}}$;
5      $t \leftarrow l_{min}$; $l_t \leftarrow l_{min}$;
6      while time step $t < |H|$ in each episode do
7          $l'_t \leftarrow \arg\max_{l'} Q((s_t, l_t), l')$;
8          if $l'_t = l_t$ then
9              Take action $a = 1$, observe $s_{t+1}, r_t$;
10             Repeatedly take action $a = 0$ for $l_{min} - 1$ times, observe $s_{t+l_{min}}$;
11             $Q((s_t, l_t), l'_t) \leftarrow Q((s_t, l_t), l'_t) + \alpha[r_t + \max_l Q((s_{t+l_{min}}, l_{min}), l) - Q((s_t, l_t), l'_t)]$;
12             $t \leftarrow t + l_{min}$; $l_t \leftarrow l_{min}$;
13         else
14             Take action $a = 0$, observe $s_{t+1}$;
15             $Q((s_t, l_t), l'_t) \leftarrow Q((s_t, l_t), l'_t) + \alpha[\max_l Q((s_{t+1}, l_t + 1), l) - Q((s_t, l_t), l'_t)]$;
16             $t \leftarrow t + 1$; $l_t \leftarrow l_t + 1$;
17 return $Q$
```

observe the last state (line 4) and update the counters (line 5). In the inner loop of iterating the time steps, we get the greedy action at the current state first (line 7). If the action is to cut the batch, we conduct action $a = 1$, record the rewards (line 9) and then conduct action $a = 0$ successively for $l_{min} - 1$ times until observing the last state (line 10). Then we update the Q-function with a l_{min}-step lookahead (line 11). Otherwise, we conduct action $a = 0$, observe the next state (line 14) and update the Q-function with a one-step lookahead (line 15).

Complexity Analysis. The space complexity of RQL is $O(C^2(l_{max} - l_{min})^2)$, *i.e.*, the memory size to store the tabular Q-function. The per-step time complexity is at most $O(H(C))$, where $H(C)$ is the time to execute the Hungarian algorithm on a bipartite graph with at most C nodes.

Other Optimization. To further reduce the memory cost to store the tabular Q-function and relieve the sparsity problem of the Q-value table, we propose quantization techniques for the RQL. The main idea is to partition both the state space and the action space into piles, and the index of each pile represents the new state or action. Intuitively, states or actions sharing similar values have similar results as the values can be considered as continuous numbers, and they can be compressed by quantization. Formally, let the quantized parameters of states and actions be $q_s \in [\![1, C]\!]$ and $q_a \in [\![1, l_{max} - l_{min}]\!]$. Suppose $s = (n_l, n_r)$, then the quantized Q-value is $Q_q = (((\frac{n_l}{q_s}, \frac{n_r}{q_s}), \frac{l}{q_a}), \frac{l'}{q_a})$. Suppose $l_q = \arg\max_{l'} Q_q(((n_l, n_r), l), l')$, then the greedy action is updated by $l_{greedy} = randint(l_q q_a, (l_q + 1)q_a - 1)$ where $randint(a, b)$ is the uniformly random function to get an integer from $[a, b]$. By applying the quantized techniques, the space complexity of RQL would be $O((\frac{C}{q_s})^2(\frac{l_{max} - l_{min}}{q_a})^2)$, which is $\frac{1}{(q_s q_a)^2}$ of the original RQL.

TABLE II: Parameter settings

Parameter	Setting
Cardinality	1K, 2.5K, **5K**, 7.5K, 10K
a_p(Power distribution)	1.1, 1.25, 1.5, 1.75, 2
a_l(Lomax distribution)	2, 3, 5, 10, 50
σ_n(Normal distribution)	0.05, 0.1, 0.15, 0.2, 0.3
α (Sparsity)	0.1, 0.3, **0.5**, 0.7, 0.9
d_{UB} (Upper bound of duration)	30, **60**, 120, 300, 600
σ_d (Variance of duration)	10, 20, 30, 40, 50
Scalability	$\|L\| = \|R\| =$ 10K, 50K, 100K

VI. Experimental Study

This section presents the experimental evaluations of our RQL algorithm on both synthetic and real datasets.

A. Experimental Setup

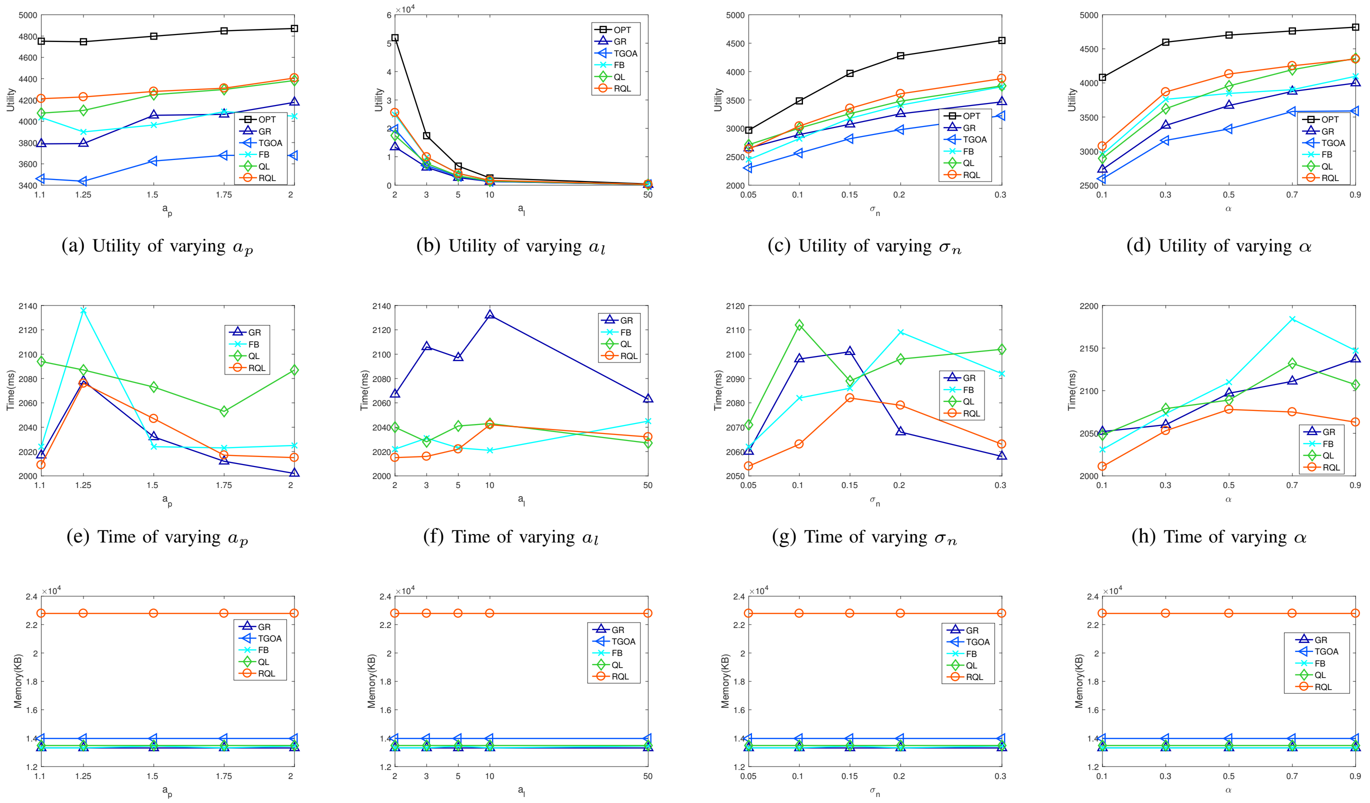
Datasets. We use both synthetic and real datasets.

The real dataset is collected by Didi Chuxing [29] in Chengdu, China, which is published through its GAIA initiative [30]. Each tuple is a taxi calling request consisting of a pickup latitude/longitude, a pickup timestamp, a drop off latitude/longitude and a drop off timestamp. We take the pickup time as the arriving time of each node on the left side and the drop off time as that on the right side. To calculatE the edge weight between each node pair, we estimate the expected revenue of each request by $d_r - d_c$ where d_r is the estimated fare of the trip and d_c is the estimated cost for the driver to pick up the passenger. If $d_c \geq d_r$, the edge will be removed. The dataset does not contain the deadline of each node, so we manually generate their durations.

For the synthetic dataset, we vary the distributions of edge weights, the sparsity of graph, the duration of nodes, the arriving density of nodes, the cardinality and the scalability. We consider three distributions, namely power distribution, lomax distribution and normal distribution, for generating the weights since they are widely used in real-world applications, *e.g.*, the income in a market economy usually follows the power distribution. The parameter settings are motivated by [31], [32] and are shown in Table II.

Compared Methods. We compared our RQL algorithm with the following algorithms.

- *Greedy algorithm (GR).* We use the greedy algorithm in Algorithm 1. Although the Greedy algorithm may perform arbitrarily bad in the worst case, it is still competitive in average efficiency [33].
- *TGOA.* It is the state-of-the-art online matching algorithm for the two-sided online maximum bipartite matching problem [4]. We modify the algorithm to our DBGM problem.
- *Fixed-batch algorithm (FB).* The algorithm uses batches with a constant size. In the evaluation, we try different



(a) Utility of varying a_p

(b) Utility of varying a_l

(c) Utility of varying σ_n

(d) Utility of varying α

(e) Time of varying a_p

(f) Time of varying a_l

(g) Time of varying σ_n

(h) Time of varying α

(i) Memory of varying a_p

(j) Memory of varying a_l

(k) Memory of varying σ_n

(l) Memory of varying α

Fig. 4: Results on varying the distribution of edge weights and the sparsity

sizes and pick the best one for each parameter setting. The in-batch matching algorithm is the Hungarian algorithm.

- *Q-learning (QL).* This is the original Q-learning algorithm [22] without the restriction principle and other optimization techniques in Sec. V-C.

Metrics and Implementation. All the algorithms are evaluated in terms of total utility score, running time (measured by milliseconds) and memory cost (measured by KB). We also add the curve of the optimal solution on the figures of utility. As TGOA always consumes 20x larger time than others, we remove the curve of TGOA when comparing the running time.

All the algorithms are implemented by C++. The experiments are conducted on Ubuntu 16.04 LTS with Intel(R) Core(TM) i7-6700 3.4GHz CPU and 16GB main memory. The experiments are repeated 10 times for each parameter setting and the average is reported.

B. Experimental Results

We have experimented the algorithms in both the remain model and the expiration model. Since the performances of the algorithms in the remain model are better than in the expiration model in most cases, we only present the results in the remain model for brevity.

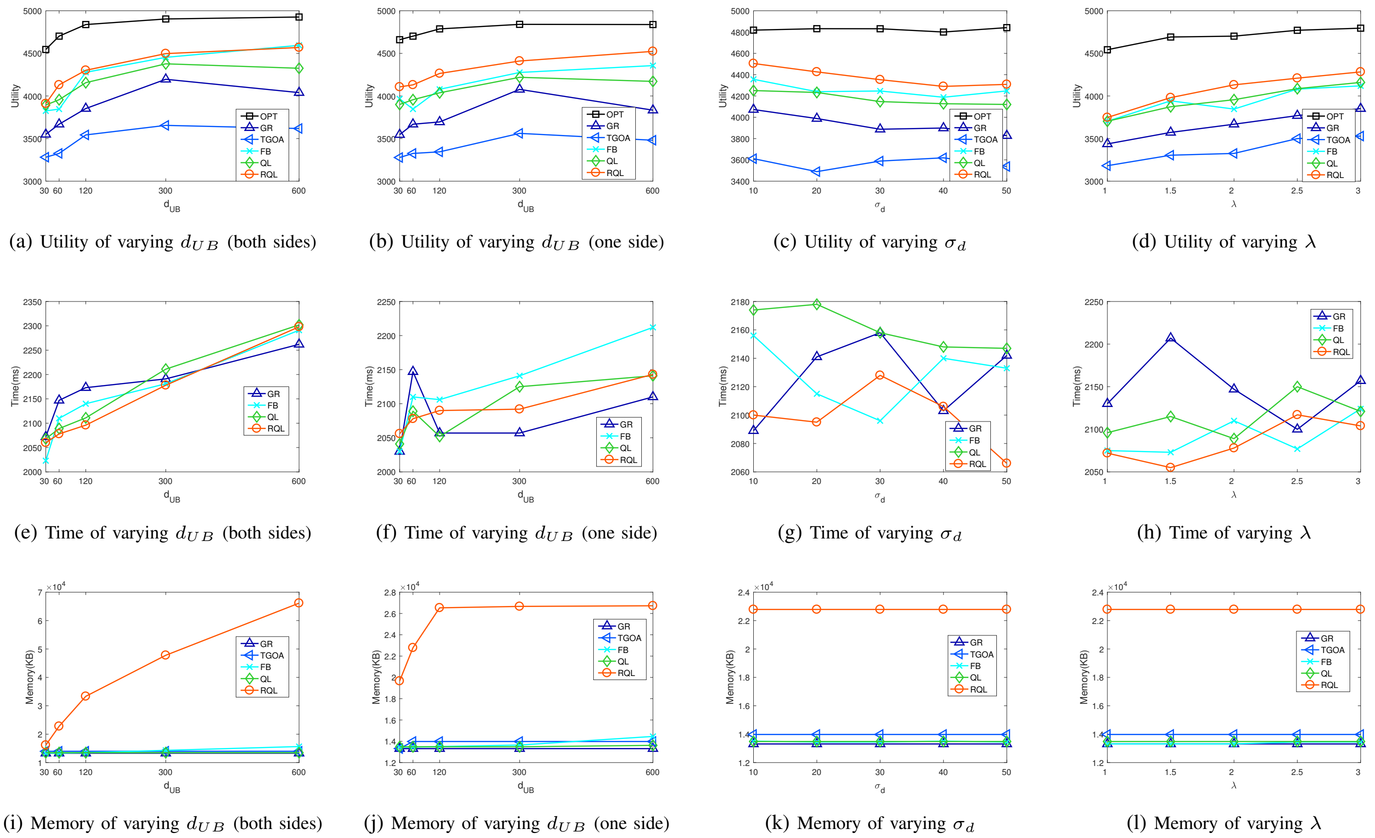
Impact of Edge Weights. This thread of experiments investigate the impact of the *distribution* and the *sparsity* of edge weights.

The first column of Fig. 4 presents the results of varying the parameter a_p of the power distribution. As a_p grows larger, the upper bound of weights becomes closer to 1. RQL performs the best, followed by QL, FB and GR while TGOA is the worst. The running time of all the algorithms is similar, but RQL always runs faster than QL, as it skips the rounds quickly when the batch size is smaller than l_{min}. The memory cost of RQL is nearly twice of others because its Q-function contains the current batch sizes as an extra slot, but the cost is acceptable. The results on memory cost are the same for the the three subsequent experiments on the impact of edge weights, so we omit the corresponding descriptions.

The second column of Fig. 4 shows the results of lomax distribution (also known as the long tail distribution). As a_l increases, the curve of its probability density function becomes more steep, leading to more small values, so the utility of each method decreases exponentially. RQL still performs the best among all the algorithms, followed by FB, QL and TGOA. GR is the worst as there are more unexpected edges with very large weights and it is better to leave the nodes waiting.

The third column of Fig. 4 shows the results of varying the variance in the normal distribution, where the mean is set to 0.5. RQL outperforms the others but the gap between RQL and QL is small. For the running time, RQL is still competitive, as it is faster than QL on all cases.

The final column of Fig. 4 shows the results of varying



(a) Utility of varying d_{UB} (both sides)

(b) Utility of varying d_{UB} (one side)

(c) Utility of varying σ_d

(d) Utility of varying λ

(e) Time of varying d_{UB} (both sides)

(f) Time of varying d_{UB} (one side)

(g) Time of varying σ_d

(h) Time of varying λ

(i) Memory of varying d_{UB} (both sides)

(j) Memory of varying d_{UB} (one side)

(k) Memory of varying σ_d

(l) Memory of varying λ

Fig. 5: Results on varying the duration and arriving density

different sparsity of the graph. The parameter α means that when we generate an edge, it has a probability of α that we remove it from the graph. When the graph grows denser, the utility gets higher. We could observe that RQL beats other methods both for utility and running time.

Impact of Duration. This series of experiments study the impact of durations of the nodes.

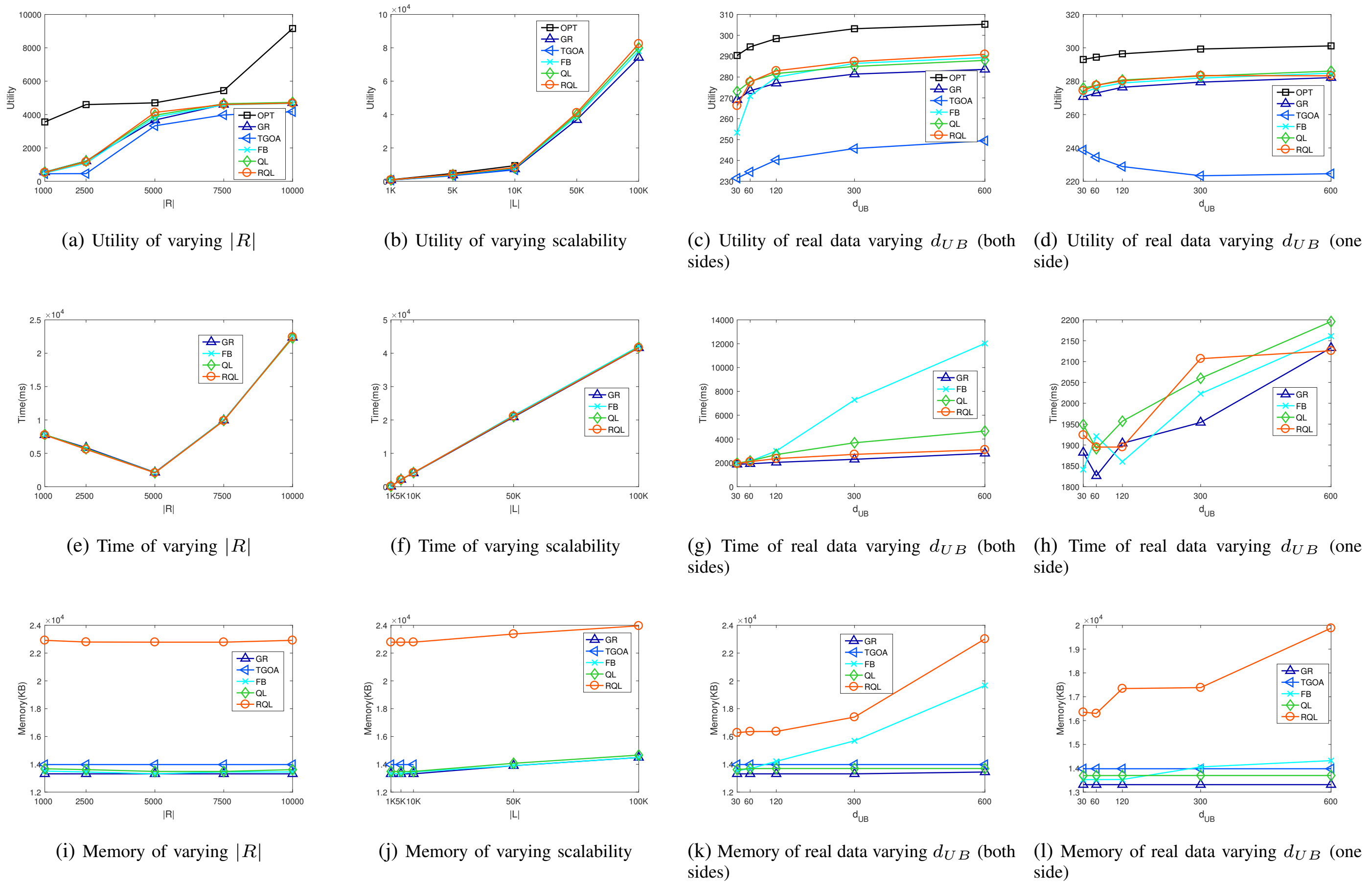
The first column of Fig. 5 shows the results of varying the upper bound of duration d_{UB} of nodes from both sides. The durations are sampled uniformly while the lower bound is set to 10. To our surprise, the utility only increases slightly with the increase of duration. It indicates that waiting for too long may not result in a much higher revenue. Also, the advantage of RQL is not obvious over others, and it is even outperformed by FB in the last case, possibly *because* the action space is too large when the duration is 600, which makes the algorithm difficult to converge. The running time also increases with the duration and RQL is still fast. However, for the memory cost, it increases with the duration, because as l_{max} grows larger, the Q-value table will consume more space.

The second column of Fig. 5 shows the results of varying d_{UB} of nodes from one side. The curve is similar to the last one, which means that the upper bound of duration does not change the performance greatly. Similar results hold for the running time. For the memory, the cost of RQL does not increase in the last two cases because the tuned value of l_{max} does not significantly increase.

The third column of Fig. 5 shows the results when we sample the duration from a truncated normal distribution with the variance from 10 to 50. The utility does not change much as the variance increases. RQL performs the best, followed by FB, QL, GR and TGOA. The results on running time and memory are similar as before.

Impact of Arriving Density. The arriving density of nodes is decided by a Poisson distribution with the parameter λ. The final column of Fig. 5 presents the results of varying λ. As λ increases, the utility also grows. The advantage of RQL is more notable with a larger λ. The results on running time and memory cost remain similar.

Impact of Cardinality. As nodes from the left side and the right side are symmetrical in the synthetic dataset, we only vary the size of $|R|$ from 1K to 10K while $|L|$ is set to 5K. The purpose is to show the results when the arriving densities of left and right sides are unbalanced. The first column in Fig. 6 shows the results. When the nodes of both sides are balanced (*i.e.*, $|R| = 5000$) RQL performs the best, but in unbalanced cases, the advantage of RQL is not obvious. The possible reason is that in the unbalance cases, the number of nodes from one side is oversized, bringing more than enough candidates to the other side. Thus it is very likely to encounter the optimal choice even with the greedy algorithm. For the running time, all the algorithms perform similarly.



(a) Utility of varying $|R|$

(b) Utility of varying scalability

(c) Utility of real data varying d_{UB} (both sides)

(d) Utility of real data varying d_{UB} (one side)

(e) Time of varying $|R|$

(f) Time of varying scalability

(g) Time of real data varying d_{UB} (both sides)

(h) Time of real data varying d_{UB} (one side)

(i) Memory of varying $|R|$

(j) Memory of varying scalability

(k) Memory of real data varying d_{UB} (both sides)

(l) Memory of real data varying d_{UB} (one side)

Fig. 6: Results on cardinality, scalability and real dataset

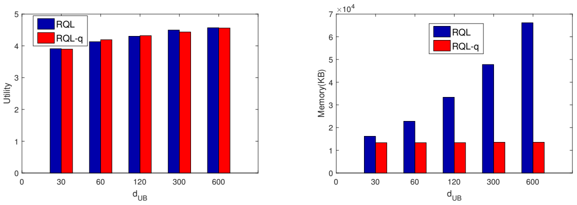
The balanced case consumes the least time, because in the unbalanced cases, all the algorithms need a waiting pool to store the nodes from the oversized side. The memory stays unchanged as the waiting pool does not consume much space, but the insertion and deletion operations will consume time.

Scalability. The the second column of Fig. 6 shows the scalability test of RQL. Due to the limitations on the time complexity for OPT and TGOA, we only plot the values of them for the first three cases. The utility increases linearly and RQL outperforms all the baselines. The running time also grows linearly, and the memory costs of all the algorithms only increase to a small extent, as they are all designed for data streams, and the expansion of data scale will not change the memory cost much.

Performance on Real Datasets. The third column of Fig. 6 presents the results of varying d_{UB} of nodes from both sides on the real dataset. The utility is similar to the first column of Fig. 5, but the advantage of RQL is not notable. The running time of FB grows the fastest, as the arriving frequency of real data varies a lot and FB has to maintain a waiting pool for a fixed time even when nodes arrive frequently. Consequently, the Hungarian algorithm is conducted on too many nodes.

The final column of Fig. 6 shows the results of varying d_{UB} of nodes from one side on the real dataset. If only the nodes from one side wait longer, the waiting pool will not be too crowded and the memory consuming problem of FB vanishes. The performance of RQL on real data is not as good as that on synthetic data. The reason is that the synthetic data are generated by stable distributions and RQL can learn some regular patterns. But for real data, the regular pattern may not be obvious, and there are also many external factors to affect the distributions. Particularly, the real data have spatial and temporal features but RQL does not make use of them as RQL is intended for general bipartite matching rather than optimized for spatial matching.

Impact of Quantization. We study the effectiveness of quantization on the most space consuming parameter setting, *i.e.*, the setting varying the upper bound of duration on both sides and where $q_s = 5$ and $q_a = 3$. The results are shown in Fig. 7, where the quantized RQL is denoted by RQL-q. In Fig. 7a, we observe that RQL-q achieves the same utility score as RQL, it even outperforms RQL in some cases, since a more compact representation of Q-value may relieve the sparsity problem and make the convergence faster. In Fig. 7b, we find that the memory cost of RQL-q remains stable, at about $15MB$ increase of duration, while that of RQL keeps mounting, which proves that applying quantization to reduce the memory cost is effective.



(a) Utility comparison

(b) Memory comparison

Fig. 7: Effect of quantized techniques

Summary of Experimental Results. In terms of utility scores, RQL is the best in most cases both on real and synthetic datasets. For the running time, all the algorithms except TGOA perform similarly, but RQL is generally faster than QL. However, the memory cost of RQL is about twice as that of the others. By applying quantization, the memory of RQL can be reduced without affecting the utility score.

VII. Conclusion

In this work, we propose a new online bipartite matching problem, the *dynamic bipartite graph matching (DBGM)* problem, which is better aligned with emerging real-world applications. On observing that most existing algorithms make real-time matching decisions which limits their performance and practicability, we propose a novel adaptive batch-based framework to solve the DBGM problem. We prove that the adaptive batch-based framework guarantees a competitive ratio of $\frac{1}{C-1}$ in the adversarial model. On basis of the framework, we devise *Restricted Q-learning (RQL)*, a reinforcement learning based algorithm that yields a near optimal batch-splitting strategy. We also propose optimization techniques to reduce the search space and the memory cost of our algorithm. Extensive experiments on both real and synthetic datasets validate the effectiveness and efficiency of our algorithm.

Acknowledgment

We are grateful to anonymous reviewers for their constructive comments. Yansheng Wang, Yongxin Tong, Ke Xu, and Weifeng Lv's works are partially supported by National Science Foundation of China (NSFC) under Grant No. 61822201, U1811463, 61532004, the Science and Technology Major Project of Beijing under Grant No. Z171100005117001, and Didi Gaia Collaborative Research Funds for Young Scholars. Cheng Long's work is partially supported by NTU SUG M4082302.020. Yongxin Tong is the corresponding author in this paper.

References

[1] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, "An optimal algorithm for on-line bipartite matching," in *STOC*, 1990, pp. 352–358.

[2] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized online matching," *Journal of the ACM (JACM)*, vol. 54, no. 5, p. 22, 2007.

[3] Z. Huang, N. Kang, Z. G. Tang, X. Wu, Y. Zhang, and X. Zhu, "How to match when all vertices arrive online," in *STOC*, 2018, pp. 17–29.

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

[5] H. Ting and X. Xiang, "Near optimal algorithms for online maximum edge-weighted b-matching and two-sided vertex-weighted b-matching," *Theor. Comput. Sci.*, vol. 607, pp. 247–256, 2015.

[6] Y. Wang and S. C. Wong, "Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm," in *ICALP*, 2015, pp. 1070–1081.

[7] Y. Tong, L. Wang, Z. Zhou, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *PVLDB*, vol. 10, no. 11, pp. 1334–1345, 2017.

[8] T. Song, Y. Tong, L. Wang, J. She, B. Yao, L. Chen, and K. Xu, "Trichromatic online matching in real-time spatial crowdsourcing," in *ICDE*, 2017, pp. 1009–1020.

[9] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou, "Latency-oriented task completion via spatial crowdsourcing," in *ICDE*, 2018, pp. 317–328.

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

[11] M. L. Littman, "Algorithms for sequential decision making," Ph.D. dissertation, 1996.

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

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998.

[14] G. Goel and A. Mehta, "Online budgeted matching in random input models with applications to adwords," in *SODA*. Society for Industrial and Applied Mathematics, 2008, pp. 982–991.

[15] J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan, "Online stochastic matching: Beating 1-1/e," in *FOCS*, 2009, pp. 117–126.

[16] B. Brubach, K. A. Sankararaman, A. Srinivasan, and P. Xu, "New algorithms, better bounds, and a novel model for online stochastic matching," in *ESA*, 2016, pp. 24:1–24:16.

[17] 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 *AAMAS*, 2018, pp. 318–326.

[18] Z. Huang, Z. G. Tang, X. Wu, and Y. Zhang, "Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals," in *ICALP*, 2018, pp. 79:1–79:14.

[19] R. I. Brafman and M. Tennenholtz, "R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

[20] M. J. Kearns and S. P. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine Learning*, vol. 49, no. 2-3, pp. 209–232, 2002.

[21] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *NIPS*, 2000, pp. 1057–1063.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[24] W. Chen, Y. Wang, and Y. Yuan, "Combinatorial multi-armed bandit: General framework and applications," in *ICML*, 2013, pp. 151–159.

[25] E. Curry *et al.*, "Adaptive task assignment in spatial crowdsourcing," Ph.D. dissertation, 2016.

[26] M. Z. Spivey and W. B. Powell, "The dynamic assignment problem," *Transportation Science*, vol. 38, no. 4, pp. 399–419, 2004.

[27] D. Gao, Y. Tong, J. She, T. Song, L. Chen, and K. Xu, "Top-k team recommendation and its variants in spatial crowdsourcing," *Data Science and Engineering*, vol. 2, no. 2, pp. 136–150, 2017.

[28] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "SLADE: A smart large-scale task decomposer in crowdsourcing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 8, pp. 1588–1601, 2018.

[29] "Didi chuxing," *https://www.didichuxing.com/*.

[30] "Gaia," *https://outreach.didichuxing.com/research/opendata/*.

[31] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *SIGMOD*, 2018, pp. 773–788.

[32] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *PVLDB*, vol. 11, no. 11, pp. 1633–1646, 2018.

[33] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis," *PVLDB*, vol. 9, no. 12, pp. 1053–1064, 2016.