Efficient Adaptive Matching for Real-Time City Express Delivery

Yafei Li, Qingshun Wu, Xin Huang, Jianliang Xu, Wanru Gao, and Mingliang Xu

Abstract—City express delivery services (a.k.a. last-mile delivery) have become more prominent in recent years. Many logistics giants, such as Amazon, JD, and Cainiao, have deployed intelligent express delivery systems to deal with the growing demand for parcel delivery. Existing works adopt queuing or batch processing approaches to assign parcels to couriers. However, these approaches do not fully consider the distribution of parcels and couriers, leading to poor quality of task assignment. In this paper, we investigate a problem of delivery matching based on revenue maximization in real-time city express delivery services. Given a set of couriers and a stream of parcel collection tasks, our problem aims to assign each collection task to a suitable courier to maximize the overall revenue of the platform. The problem is shown to be NP-hard. To tackle the problem efficiently, we present a time-aware batch matching algorithm to offer high-quality courier-task matching in each sliding window. We further theoretically analyze the matching approximation bound. In addition, we propose an efficient deep reinforcement learning based approach to adaptively determine the sliding window size for better matching results. Finally, extensive experiments demonstrate that our proposed algorithms can achieve desirable effectiveness and efficiency under a wide range of parameter settings.

Index Terms—City express delivery, location-based service, real-time system, optimization, adaptive matching

1 Introduction

Driven by the continuous proliferation of e-commerce and logistics industries, city express delivery services (*a.k.a.* last-mile delivery) have become more prominent in recent years. As shown by the latest statistics [1], in 2020, the business volume of city express delivery services in China has exceeded a total of 70 billion transactions and achieved an increase of 12.6% over 2019. Many logistics giants such as Amazon [2], JD [3], and Cainiao [4], have deployed intelligent real-time express delivery services to deal with the huge demand. In a typical city express delivery system, the platform manages a number of express stations (hereinafter referred to simply as *stations*) to provide parcel delivery and collection services for the whole city. The platform distributes the parcels to the relevant stations whose service regions can cover the parcels' destinations. In practice, each station typically receives the parcels to be delivered twice a day (*e.g.*, at 8:00 am and 1:00 pm). After receiving the parcels, the station immediately arranges suitable couriers to deliver them to the customers. On guaranteeing the time constraints of parcel deliveries, the platform dynamically arranges couriers to collect online ad-hoc parcel collections requested by customers. Compared with the traditional way of separating the parcel delivery and collection services, a courier in a real-time city express delivery service can complete parcel collections on the way to delivering parcels, which significantly improves the efficiency of the service. In this paper, we focus on efficiently allocating parcel collections to couriers while guaranteeing the time and capacity constraints of parcel deliveries. Fig. 1 illustrates the general process of a real-time city express delivery service.

Example 1. In Fig. 1, there exists a station s_1 and two couriers c_1 and c_2. At first, c_1 loads parcels d_1, d_2, d_3 and c_2 loads the parcels d_4, d_5, d_6 for delivery according to individual routes (*i.e.*, c_1 the red solid line and c_2 the blue solid line). Assume now the platform receives two parcel collection requests r_1 and r_2 when c_1 and c_2 leave for the destinations of d_1 and d_4. Since r_1 is near the route of c_1 and r_2 is near the route of c_2, the platform arranges for c_1 to collect r_1 and c_2 to collect r_2. Otherwise, c_1 and c_2 may not complete their parcel deliveries (i.e., delivering all parcels and returning to the station in time).

The city express delivery problem is essentially a task assignment problem. Current works related to task assignment can be categorized into two types: i) *instant assignment* [5], [6], [7], [8], the system assigns each task to a suitable courier immediately when it arrives. If there is no courier who can serve the task, it will wait until a qualified courier appears or the task's deadline is passed; ii) *batch assignment* [9], [10], [11], the system periodically fetches the tasks from a task queue in batches, matches them with currently available couriers, and finds the best assignments between tasks and couriers. For the tasks that are not assigned in the current

- *Yafei Li, Qingshun Wu, Wanru Gao, and Mingliang Xu are with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, Henan 450001, China. E-mail: yafeics@outlook.com, wqszzu@gs.zzu.edu.cn, {iewrgao, iexumingliang}@zzu.edu.cn.*
- *Xin Huang and Jianliang Xu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, China. E-mail: {xinhuang, xujl}@comp.hkbu.edu.hk.*

Manuscript received 7 Sept. 2021; revised 15 Feb. 2022; accepted 19 Mar. 2022. Date of publication 24 Mar. 2022; date of current version 1 May 2023.
This work was supported in part by the NSFC under Grants 61972362, 62036010, 61822701, 62106231, and 61602420, in part by CPSF under Grant 2018M630836, in part by HNSF under Grant 202300410378, in part by Hong Kong RGC under Grants 12200021 and C2004-21GF, and in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2019B1515130001. (Corresponding author: Mingliang Xu.)
Recommended for acceptance by M. A. Cheema.
Digital Object Identifier no. 10.1109/TKDE.2022.3162220

Fig. 1. A toy example of real-time city express delivery.

batch, they will be assigned again in the next batch. However, both of these two approaches have issues. With regard to the instant assignment approach, it is easy to implement but may not generate high quality matching results [12], [13]. While the batch assignment approach can improve the quality of matching results to some extent, manually tuning the batch size to achieve a better result is not easy in real-time scenarios. In addition, optimizing the revenue of the platform is one of the main concerns for most commercial platforms. However, existing studies mainly focus on improving the computational efficiency rather than increasing the platform revenue.

In this paper, we investigate a novel problem of real-time city express delivery via an adaptive sliding window (denoted as RTDW). In the RTDW problem, a stream of parcel collections enters the platform in real time. The platform dynamically dispatches the most suitable parcel collections to couriers by considering several constraints (*e.g.*, time, capacity, and detour) and aims to maximize the platform's revenue. However, due to the parcel collections stochastically appearing in the platform, the dispatch processing is highly dynamic. As such, there are two key challenges to be resolved for the RTDW problem. First, considering a set of couriers and a set of parcel collection tasks, the issue of how to efficiently match them with a quality guarantee is hard to resolve. Second, as well as being widely used in many existing works, sliding window is a common processing strategy in real-time scenarios; however, dynamically determining the sliding window size to achieve a better matching result is still a challenge. To address the above two issues, we present an efficient time-aware batch matching (TBM) algorithm for the first issue which guarantees the matching result under a 2-approximation bound. For the latter issue, on the basis of TBM, we propose an efficient deep reinforcement learning (DRL) based optimization that can adaptively determine the sliding window size for better matching results. The main contributions of this paper can be summarized as follows:

- We first present an adaptive matching framework of city express delivery, equipped with three entries of delivery tasks, collection tasks, and couriers. Based on the constraints of courier schedule, we formally define a novel RTDW problem which aims to maximize the platform revenue by assigning each courier with suitable collection tasks.
- We present two efficient algorithms SMA and TBM to solve the RTDW problem. SMA is a greedy approach for fast finding the suitable courier for each collection task. Furthermore, we develop an effective partition strategy for sliding window based on the distribution of historical collection tasks. Integrating this partition strategy, we propose an efficient algorithm TBM to find the better matching result in a sliding window under a 2-approximation bound on quality.
- We further present an efficient DRL-based optimization that is equipped with a novel state representation to adaptively determine the sliding window size for a good long-term revenue of the platform. In addition, we also theoretically analyze the competition ratios of our proposed algorithms.
- We conduct extensive experiments to demonstrate that our proposed algorithms can achieve desirable effectiveness and efficiency under different parameter settings.

TABLE 1
Notations and Descriptions

Notations	Description
G	a road network
Γ	a set of delivery tasks
Λ	a set of collection tasks
γ	a delivery task
λ	a collection task
t_γ	the deadline of the delivery task
t_λ	the deadline of the collection task
C	a set of couriers
k_c	the maximum capacity of courier c
$R(c, \lambda)$	the revenue of a courier c collecting λ
$\mathbb{E}(\mathcal{M})$	the total revenue of a matching plan $\mathcal{M}$
w	a sliding window
$\mathcal{M}$	a feasible matching allocation
$\mathcal{M}_w$	a feasible matching of sliding window w
h	a time slice

The remainder of this paper is organized as follows. We first formulate the RTDW problem in Section 2. Then, in Section 3, we propose several solutions to solve the RTDW problem. Next, we evaluate our solutions in Section 4 and review the related work in Section 5. Finally, in Section 6, we conclude the paper and introduce several possible optimizations for future works.

2 Problem Formulation

In this section, we present several preliminaries and provide the problem statement, followed by a theorem to establish the hardness of the RTDW problem. Table 1 summarizes the notations frequently used in this paper.

2.1 Basic Concepts

In our RTDW problem, we consider three entities that consist of a set of stations S, a set of couriers C, and a road network $G = (L, E, W)$. For road network G, each vertex $l \in L$ denotes a road intersection and each edge $e_{ij} \in E$ denotes a road segment with a weight $w_{ij} \in W$ linking two road intersections l_i and l_j. The platform manages a number of courier

stations S to provide the whole city's express service. Each station $s \in S$ is represented as a tuple $s = (l_s, C_s, R_s)$, where $l_s \in L$ is the location of s, C_s is the set of couriers hired in station s, and R_s is the service region of s. Following the settings of existing works [14], [15] and commercial platforms [3], [4], in this paper each station manages a number of couriers to deliver and collect parcels in its service region. Next, we give the definitions of delivery task and collection task.

Definition 1 (Delivery task). *A delivery task is a two-entry tuple $\gamma = (l_\gamma, t_\gamma)$ indicating that the parcel taken from the station should be delivered to location l_γ before the deadline t_γ.*

Definition 2 (Collection task). *A collection task is a four-entry tuple $\lambda = (b_\lambda, l_\lambda, t_\lambda, \rho_\lambda)$ indicating that the parcel appearing at time b_λ should be collected at location l_λ before the deadline t_λ, and the corresponding fare is ρ_λ.*

We clarify that the definitions of delivery task and collection task are derived from real express delivery services. On one hand, each courier is fully loaded with delivery tasks and starts from the station to deliver these parcels to the locations of customers on or before their delivery deadlines. Note that the courier usually earns a monthly basic salary by completing the daily delivery tasks assigned by the station; On the other hand, the platform usually encourages couriers to complete as many collection tasks as possible and that cuts a fixed rate of revenue from each completed collection task as the payment of the courier, because the number of collection tasks usually determines the amount of platform revenue. According to the definitions of delivery task and collection task, we can define a courier as follows:

Definition 3 (Courier). *A courier $c \in C$ is denoted as a six-entry tuple $c = (l_c, s_c, k_c, t_c, \Gamma_c, \Lambda_c)$, where l_c is the current location of c, s_c is the belonging station, k_c is the maximum capacity, t_c is the deadline of needing to be back at s_c, Γ_c is a set of delivery tasks, and Λ_c is a set of collection tasks.*

We remark that couriers normally receive several batches of delivery tasks per day from the stations they belong to. A courier should complete its own current batch of delivery tasks and return to its own station before the next batch starts. The platform dynamically arranges couriers to complete the collection tasks that arrive in a stream. Note that each courier c fully loads a set of delivery tasks Γ_c ($|\Gamma_c| = k_c$) when leaving the station and will be allocated with some collection tasks Λ_c while executing delivery tasks under the constraints that the tasks in Γ_c and Λ_c should be finished on or before t_c.

Definition 4 (Courier Schedule). *Given a courier $c \in C$, the schedule of c denoted as $\mathcal{S}_c = \langle l_1, l_2, \ldots, l_m \rangle$ is a sequence of locations, where each location $l_k \in L$ is to be reached by c to execute a collection task of Λ_c or a delivery task of Γ_c.*

A schedule $\mathcal{S}_c$ is valid iff for any collection/delivery task assigned to courier c, courier c can satisfy their deadline constraints, and the number of collection and delivery tasks left cannot exceed its maximum capacity k_c at any time. Here, we assume that couriers always select the shortest path between two points. Moreover, if a collection task λ can be served by a courier c without violating the time and capacity constraints, c and λ are a valid courier-task pair $\langle c, \lambda \rangle$. Following the existing works [16], [17], [18], in this paper we assume that the order of the existing tasks in the schedule $\mathcal{S}_c$ remains unchanged when inserting a new collection task λ into $\mathcal{S}_c$, and λ will be inserted between two consecutive tasks in $\mathcal{S}_c$ that incurs the smallest detour distance. Meanwhile, the time constraints of the new collection task λ and the tasks in $\mathcal{S}_c$ after λ should be satisfied. Once any time constraint is broken, we consider the new collection λ cannot be served by the courier c.

Next, we define the revenue brought to the platform by a courier c serving a set of collection tasks Λ_c as follows.

$$C_R(c, \Lambda_c) = \rho_{\Lambda_c} - \kappa \cdot \pi(c, \Lambda_c), \tag{1}$$

where ρ_{Λ_c} is the sum of the fare of $\lambda \in \Lambda_c$, κ is the cost per kilometer (*i.e.*, fees paid to couriers), and $\pi(c, \Lambda_c)$ is the detour cost of c finishing Λ_c (*i.e.*, the travel distance difference between collecting Λ_c and not collecting Λ_c). Note that the above revenue model is widely adopted in other transport management systems, such as ridesharing [19], [20] and spatial crowdsourcing [21].

On the basis of the above definitions, we can formally formulate our studied RTDW problem as follows:

Definition 5 (RTDW Problem). *Given a set of couriers C and a stream of collection tasks Λ, the RTDW problem is to find the best matching plan $\mathcal{M} \subseteq C \times \Lambda$ such that the total platform revenue $\mathbb{E}(\mathcal{M})$ is maximized,*

$$\mathbb{E}(\mathcal{M}) = \sum_{c \in C, \Lambda_c \subseteq \Lambda} C_R(c, \Lambda_c), \tag{2}$$

where for each matching $(c, \Lambda_c) \in \mathcal{M}$, c and Λ_c should satisfy two conditions: i) each $\lambda \in \Lambda_c$ should be collected by c on or before its deadlines t_λ; ii) the sum of $|\Gamma_c|$ and $|\Lambda_c|$ should not exceed k_c at any time.

2.2 Problem Hardness

We theoretically analyze the hardness of the RTDW problem.

Theorem 1. *The RTDW problem is NP-hard.*

Proof. We omit the proof due to the space limitation and put it in Appendix A.1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2022.3162220. □

Discussion. Ideally, the objective of the RTDW problem is to maximize the platform's total revenue over the collection tasks in the entire stream and all the available couriers. In reality, considering the rigid real-time scenario, the platform only holds the information of current collection tasks and couriers and thus is impossible to achieve the highest revenue over the entire collection task stream. In addition, as proved in Theorem 1, the RTDW problem is NP-hard. As such, similar to the greedy approach used in many real-time streaming data processing problems, in this paper we adopt the approaches of sequential processing or batch processing to solve the RTDW problem.

3 Proposed Solutions

In this section, we present a novel two-stage solution framework that consists of batch division and batch matching for

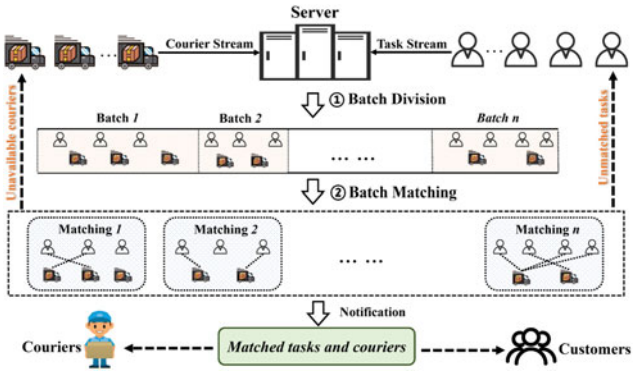


Fig. 2. The solution framework of the RTDW problem.

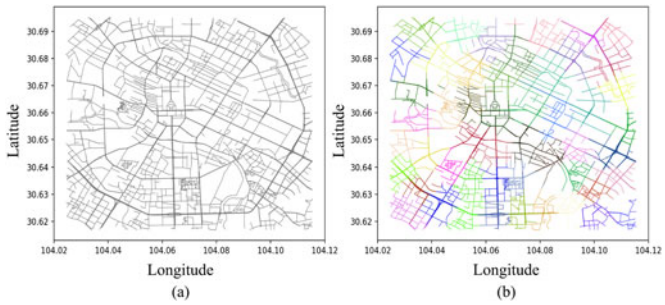


Fig. 3. An example of service region division.

solving the RTDW problem. Based on this solution framework, we present three courier-task matching solutions. In what follows, we introduce them one by one in details.

3.1 Solution Framework

To solve the RTDW problem, we propose a novel solution framework that consists of batch division and batch matching, as illustrated in Fig. 2. Initially, the customers send in their requested collection tasks to the server, while the couriers report their current statuses (e.g., location, capacity, and schedule) to the server. After receiving the collection tasks, the server divides the collection tasks into a sequence of batches to match based on some division strategies (Step-1). According to batch's sequence, the server conducts the matching for the collection tasks and available couriers in a batch to maximize the revenue of the platform, while the available couriers and unmatched collection tasks return to the streams for the next batch matching, and the expired collection tasks will be discarded from the server (Step-2). Finally, the matching results are notified to the couriers and customers, respectively.

3.2 Service Region Division

As discussed in Section 2, the RTDW problem is NP-hard, it is very challenging to make real-time assignment for a huge number of couriers and collection tasks in a city per day. To reduce the computing complexity, we divide the whole city into a set of disjoint service regions and focus on each of the service regions separately. Such division strategy also brings practical benefits. For example, station managers often prefer managing the business in their own service regions separately, which is consistent with the settings of commercial platforms in reality [14], [15].

In this paper, we divide the road network into a set of service regions by the *K-means* algorithm that is one of the most frequently used clustering algorithms. More specifically, we use the *K-means* algorithm to classify the vertices of the road network into k independent spatial clusters according to their geographical distribution. Correspondingly, the whole city is divided into k independent service regions where any two service regions do not overlap with each other. As illustrated in Fig. 3, Fig. 3a shows the road network of Chengdu, Fig. 3b shows the service regions of Chengdu divided by *K-means* algorithm where each color represents a service region. Based on the divided service regions, we set the station at the center of each service region. Specifically, for each service region, we select the vertex with the smallest sum of distances to other vertices as the center of the service region. Note that the service region division requires a one-time offline calculation, it will not affect the efficiency of online assignment.

3.3 Sequential Matching Solution

As mentioned in our solution framework, the server divides the stream of collection tasks into batches according to a certain strategy and then performs batch-by-batch matching. In this section, we present a sequential matching algorithm (SMA) to solve the RTDW problem where the collection task is immediately matched when it arrives in the server. The main idea of SMA is illustrated as Algorithm 1. Given a collection task λ and a set of couriers C, we first find out the couriers C' who can serve the collection task λ without violating the deadline and capacity constraints (lines 2-4). Then, for the couriers C' we find, we return the nearest courier $c \in C'$ to the collection task λ as the result λ (line 5).

Algorithm 1. SMA Algorithm

Input: A collection task λ and a set of couriers C
Output: A feasible courier c
1: $C' \leftarrow \phi$
2: **foreach** courier $c' \in C$ **do**
3: **if** c' can serve λ **then**
4: Add c' into C'
5: Match the courier in $c \in C'$ nearest to λ, i.e., $c \leftarrow \arg\min_{c \in C'} |l_c - l_\lambda|$;
6: **return** c;

Complexity analysis. The SMA algorithm takes $O(|C|)$ time to deal with a collection λ. Thus, considering a stream of collection tasks Λ, the time complexity to finish courier-task matching for all the collection tasks Λ is $O(|\Lambda||C|)$. The SMA algorithm is simple and fast, especially for a large-scale RTDW problem, but the quality of the result obtained by this approach is not that good.

3.4 Time-Aware Batch Matching Solution

The SMA algorithm actually treats each collection task in the stream as a batch, it does not consider the batch division based on the distribution of collection tasks, leading to a poor matching result. In this section, we propose an efficient *time-aware batch matching (TBM)* algorithm to solve the RTDW problem. We adopt sliding window to divide the collection task stream into batches of reasonable size to match, which improves the quality of the solution against the SMA algorithm. Next, we start with a definition of sliding window.

Definition 6 (Sliding Window). *A sliding window is denoted by $w = \{h_0, h_1, \ldots, h_i, \ldots, h_b | b \in \mathbb{N}\}$, where h_i is a time slice and each time slice appears at most once.*

A good sliding window division strategy is extremely useful to improve overall matching quality. Inspired by this, we propose a novel sliding window division method that is implemented by the DBSCAN algorithm [22] in an offline model. Simply put, the whole day is divided into a sequence of sliding windows by the DBSCAN algorithm in terms of the deadline distribution of historical tasks. After dividing the sliding windows, we split the collection task stream into a set of batches based on the divided sliding windows. Then, we invoke Algorithm 2 to find the suitable couriers for each batch of collection tasks. For the unassigned collection tasks in the current batch, we will assign the suitable couriers for them in the next batch. Compared with existing equal-size sliding window processing methods [15], our method makes good use of the distribution information of historical orders and can determine the size of collection task batch more reasonably. Leveraging the divided sliding windows, for each sliding window, we use a group-based greedy strategy to match the couriers and collection tasks iteratively, making sure the quality of the matching is under a 2-approximation bound. Note that the sliding window division is processed in the offline model, which does not take up the online processing time.

Algorithm 2. TBM Algorithm

Input: A set of collection tasks Λ_w and a set of couriers C_w in the sliding window w
Output: A feasible matching $\mathcal{M}_w$ of w

1: A feasible matching $\mathcal{M}_w \leftarrow \phi$;
2: A set of collection task groups $\mathcal{G} \leftarrow \phi$;
3: **for** $k \leftarrow 1$ to δ **do**
4: $\quad \overline{\mathcal{G}} \leftarrow$ the set of collection task groups containing k different collection tasks in Λ_w;
5: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup \overline{\mathcal{G}}$;
6: **repeat**
7: $\quad$ **foreach** courier $c \in C_w$ **do**
8: $\quad\quad g^* \leftarrow$ the collection task group in $\mathcal{G}$ with the highest revenue that c can serve;
9: $\quad\quad$ **if** c is the best courier for g^* **then**
10: $\quad\quad\quad \mathcal{M}_w \leftarrow \mathcal{M}_w \cup \bigcup_{\lambda \in g^*} \{\langle c, \lambda \rangle\}$;
11: $\quad\quad\quad$ Remove the task group $g \in \mathcal{G}$ that contains any collection task $\lambda \in g^*$;
12: **until** none task group can be assigned;
13: **return** $\mathcal{M}_w$;

Algorithm 2 presents the procedures of our TBM algorithm. First, we initialize a matching set $\mathcal{M}_w$ and a set of collection task groups $\mathcal{G}$ (lines 1-2). Then, we divide the collection tasks into a set of groups with a size less than δ by considering their time constraints (lines 3-5). Specifically, the process of task grouping is that of enumerating all size-k task groups and checks whether they are valid. For a systematic enumeration of all candidate task groups, we employ the branch and bound algorithm presented in [23]. During the grouping process, we present an efficient rule to accelerate the pruning process, that is, two collection tasks can be grouped if and only if there exists a nearby courier who can serve them together based on the constraints of location, time, and capacity. As for the group size δ, it is set based on user historical experience. In practice, the value of δ is usually small due to the location, time, and capacity constraints. After that, we select one or more suitable task group for each courier $c \in C_w$ in a multi-round approach (lines 6-11). In each round of assignment, for each courier $c \in C_w$, if there exists a collection task group $g* \in \mathcal{G}$ such that $g*$ is the best for c and c is the best for $g*$, we assign $g*$ to c and remove the task groups including the task $\lambda \in g*$ from $\mathcal{G}$. Note that there is at least one task group in $\mathcal{G}$ assigned to a courier in each round of assignment, and the multi-round assignments will stop until none of the task groups in $\mathcal{G}$ can match a suitable courier in a round. In what follows, we illustrates the TBM algorithm by a running example as follows.

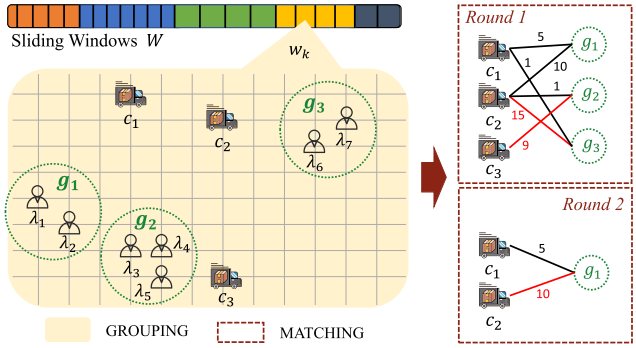


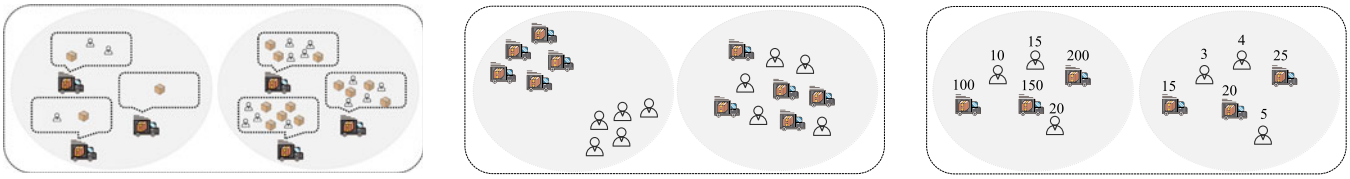
Fig. 4. A running example for the TBM algorithm. The yellow area shows the distribution of couriers $c_1 \sim c_3$ and collection tasks $\lambda_1 \sim \lambda_7$ as well as the task groups $g_1 \sim g_3$. The dotted rectangle area shows two rounds of assignments between couriers $c_1 \sim c_3$ and task groups $g_1 \sim g_3$.

Example 2. In Fig. 4, Algorithm 2 is invoked iteratively to find the optimal assignment of couriers and tasks in each sliding window. Taking the k-th sliding window w_k as an example, we observe that there exist three couriers $c_1 \sim c_3$, and seven collection tasks $\lambda_1 \sim \lambda_7$ in w_k. Assume the collection tasks $\lambda_1 \sim \lambda_7$ can form three task groups $g_1 \sim g_3$ by the grouping method stated above. Then, we perform multiple rounds of assignments between couriers and collection task groups according to the assigning strategy illustrated in Algorithm 2. In each round of assignment, only couriers and task groups that best suit each other can be assigned. As shown in Fig. 4, in the first round of assignment (e.g., Round 1 in Fig. 4), c_2 and g_3 (c_3 and g_2) are mutually optimal. Then, g_3 and g_2 are assigned to c_2 and c_3, respectively. Meanwhile, g_2 and g_3 are removed from the system. In the second round of assignment (e.g., Round 2 in Fig. 4), g_1 is assigned to c_2 since c_2 and g_1 are the best for each other, and then g_1 is removed from the system. Finally, since there is no task group left, the round assignment is terminated.

The approximation ratio of Algorithm 2 is proved in Theorem 2.

Theorem 2. *Given a set of collection tasks Λ_w and a set of couriers C_w, let $\mathcal{M}_w$ be the matching derived by Algorithm 2 and $\mathcal{M}_w^*$ be the optimal matching, $\mathbb{E}(\mathcal{M}_w) \geq \frac{1}{2}\mathbb{E}(\mathcal{M}_w^*)$ holds.*

Proof. We omit the proof due to the space limitation and put it in Appendix A.2, available in the online supplemental material. □



(a) **Case 1:** unoccupied capacity (b) **Case 2:** distribution (c) **Case 3:** available time

Fig. 5. Illustration of state representation.

Complexity analysis. The time complexity of the TBM algorithm is $O(|\Lambda_w|^\delta + |C_w||\mathcal{G}|)$, where $|\Lambda_w|^\delta$ is the time cost of collection group division (lines 3–5) and $|C_w||\mathcal{G}|$ is the time cost of group assignment (lines 6–11).

Discussion. The TBM algorithm employs an effective partition strategy to divide the sliding window based on the distribution of historical collection tasks and that achieves a 2-approximation bound on quality for the matchings between couriers and collection tasks in each sliding window. However, such clustering-based sliding window partition strategy cannot make full use of the supply-demand relationship of couriers and collection tasks, which may affect the overall quality of the results.

3.5 DRL-Based Solution

The sliding window partition strategy we present in TBM algorithm improves the quality of task assignment, however, we observe some counterintuitive cases in practical settings, *e.g.*, the collection tasks and couriers in a sliding window cannot achieve any matching due to the insufficient unoccupied capacity. Motivated by this observation, in this section, we use the model of Markov decision process model (MDP) [24] with unknown parameters to model the sliding window decision process and adopt the DRL-based method to adaptively determine the sliding window size in terms of several combined features.

Given an RTDW problem $\mathcal{P}$, we formulate a Markov decision process $M = (S, \mathcal{A}, R^a_{ss'}, P^a_{ss'}, \xi)$ on $\mathcal{P}$, where

- S is the state space, where each state $s \in S$ denotes a bipartite graph composed of current couriers and collection tasks;
- $\mathcal{A} = [h_{min}, h_{max}]$ denotes the action space where h_{min}, $h_{max} \in \mathbb{N}$ are the minimum and maximum time slices to divide the sliding window respectively. Each action $a \in \mathcal{A}$ means the time slice to perform the best matching for the accumulated collection tasks (*i.e.*, invoke the TBM algorithm);
- $R^a_{ss'} : S \times \mathcal{A} \times S \to \mathbb{R}$ is the reward function. If the states transit from s to s' by executing the action a, we will obtain a reward of $R^a_{ss'}$ (*i.e.*, $\mathbb{E}(\mathcal{M})$);
- $P^a_{ss'} : S \times \mathcal{A} \times S \to [0, 1]$ denotes the probability of transiting the states from s to s' by executing the action a, which models the dynamic process for couriers and collection tasks entering or leaving the platform;
- $\xi \in [0, 1]$ is the discount factor.

To better determine the sliding window size, we present a novel state representation. Given a set of couriers C_h and a set of collection tasks Λ_h at time slice h, we denote each state as a feature vector $s_h = (|\mathcal{L}|, |\mathcal{R}|, |\mathcal{F}|, |\mathcal{S}|, |\mathcal{T}|, h)$, where $|\mathcal{L}|$ is the number of couriers, $|\mathcal{R}|$ is the number of collection tasks, and $|\mathcal{F}|$ is the total remaining capacity of all couriers:

$$|\mathcal{F}| = \sum_{c \in C_h} (k_c - |\Gamma_c| - |\Lambda_c|), \tag{3}$$

where $|\Gamma_c|$ and $|\Lambda_c|$ denote the numbers of delivery tasks and collection tasks of courier c, respectively. Fig. 5a illustrates the state of unoccupied capacity. We can observe that the total unoccupied capacity of the couriers on the left side of Fig. 5a is sufficient. Therefore, these couriers can wait for more collection tasks to match. For the couriers on the right side of Fig. 5a, since the unoccupied capacities of couriers cannot accept more collection tasks, it is suitable to divide the sliding window for matching. $|\mathcal{S}|$ is the distribution metric for the couriers and collection tasks:

$$|\mathcal{S}| = \sum_{\lambda \in \Lambda_h} D_M(l_\lambda, L_d). \tag{4}$$

Here, $D_M(l_\lambda, L_d)$ is the Mahalanobis distance [25], where l_λ is the location of collection task λ and L_d is the location distribution of couriers. Typically, the smaller the value $|S|$, the closer the couriers are to the parcels. This situation is suitable for dividing the sliding window. On the left side of Fig. 5b, the distribution of couriers and collection tasks are scattered. Hence, the matching may not achieve good performance at this time. In contrast, the distribution of the couriers and collection tasks on the right side of Fig. 5b is evenly distributed and is suitable for dividing the sliding window. $|\mathcal{T}|$ is the time metric to measure the general gap between the current time slice and the deadline of collection tasks:

$$|\mathcal{T}| = \min_{x \in \{t_c - \tau | c \in C_h\} \cup \{t_\lambda - \tau | \lambda \in \Lambda_h\}} x, \tag{5}$$

where τ is the current time slice. If $|\mathcal{T}|$ is smaller, it means that there exist couriers and collection tasks close to their deadlines and it is suitable to divide the sliding window at the current time. For example, comparing the two cases in Fig. 5c, the case on the right side is suitable for dividing the sliding window at once, since there exist a courier and a collection task whose deadlines are almost there.

For an action $a \in \mathcal{A}$, we represent it by a value between h_{min} and h_{max}, since making the size of the sliding window too large or too small is not an optimal strategy in practice. If the sliding window is set too small, there are less collection tasks participating in matching, leading to a poor matching result; otherwise, too many collection tasks contained in a sliding window may cause a huge computational cost and cannot meet the needs of real-time scenarios. In this paper, we set h_{min} and h_{max} on the basis of historical

collection tasks. In detail, we cluster historical collection tasks according to the distribution of arrival time, and then select the lower and upper bounds of the time interval in length concentration as h_{min} and h_{max} respectively. For the action selection, our DRL algorithm selects the value of action a from the time interval $[h_{min}, h_{max}]$ with the highest revenue based on the current state s_h and the previous learning strategy.

After modeling the sliding window decision process, we present a DRL-based algorithm to find an optimal strategy for dividing the sliding window, and the pseudo code is illustrated in Algorithm 3. We first perform a series of initializations (lines 1-4). Specifically, for Q-network Q and its target network $\hat{Q}$, we use the same random weights θ and θ^- for initialization (line 1). Moreover, we initialize the action space with h_{min} and h_{max}. Then, we iteratively collect experience and update the network (*i.e.*, learning) (lines 5-16). For action a_h, the probability of ϵ is determined by the network, and the probability of $1-\epsilon$ is randomly selected in the action space (line 7). We store every "trial and error" (*i.e.*, the process of performing action a_h) experience φ_h in the experience memory Ω (lines 8-9). For network updates (lines 10-14), we first randomly sample a batch of experiences Ψ from Ω (line 10). Then, we calculate the target value (line 12), perform a gradient descent (line 13), and calculate the back-propagation error (line 14). Finally, after every N steps, we synchronize the parameters of the Q network to its target network $\hat{Q}$ (line 16).

Algorithm 3. DRL-based Algorithm

Input: greedy factor $\epsilon \in [0,1]$, discount factor $\xi \in [0,1]$, learning rate $\alpha \in [0,1]$, iteration step E, h_{min}, h_{max}, synchronous step N

Output: memory Ω

1: Initialize the Q-network Q with random weights θ and its target network $\hat{Q}$ with random weights $\theta^- = \theta$
2: Initialize experience replay memory Ω
3: Initialize action space $\mathcal{A} = [h_{min}, h_{max}]$
4: Initialize the initial state s_0 with random action $a_0 \in \mathcal{A}$
5: **for** episode $\leftarrow 1$ to E **do**
6: **repeat**
7: $$a_h = \begin{cases} \arg\max_{a \in \mathcal{A}} \hat{Q}(s_h, a, \theta^-), & with\ \epsilon \\ random\ a \in \mathcal{A}, & with\ 1-\epsilon \end{cases} \quad ()$$
8: execute action a_h, obtain reward r_h and next state s_{h+1}
9: store experience $\varphi_h = (s_h, a_h, r_h, s_{h+1})$ in memory Ω
10: random sampling $\Psi = \{\varphi_1, \varphi_2, \ldots, \varphi_I\}$ from Ω
11: **foreach** experience $\varphi \in \Psi$ **do**
12: calculate target value $y_i = r_i + \xi \max_{a_{i+1} \in \mathcal{A}} \hat{Q}(s_{i+1}, a_{i+1}, \theta^-)$, with $\hat{Q}(s_{i+1}, a_{i+1}, \theta^-) = 0$ if s_{i+1} is terminal
13: perform a gradient descent step on $(y_i - Q(s_i, a_i, \theta))^2$ with respect to the network parameters θ
14: $\Delta\theta = \alpha(r_i + \xi \max_{a_{i+1} \in \mathcal{A}} \hat{Q}(s_{i+1}, a_{i+1}, \theta^-) - Q(s_i, a_i, \theta)) \nabla_\theta Q(s_i, a_i, \theta)$
15: **until** s_{h+1} is terminal;
16: $\hat{Q} \leftarrow Q$ every N steps
17: **return** memory Ω;

Fig. 6 illustrates the sliding window division framework that consists of planning, learning, and their interaction with the environment. The workflow of the sliding window

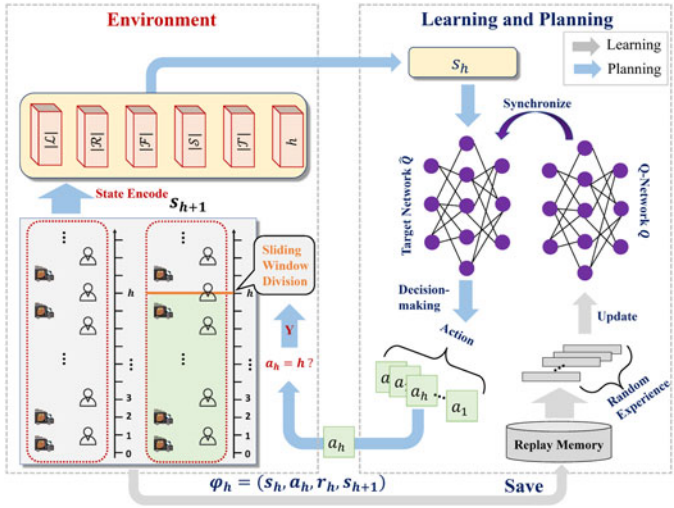


Fig. 6. The framework of the DRL-based solution.

division can be expressed as follows. First, we encode the environment state as a feature vector $s_h = (|\mathcal{L}|, |\mathcal{R}|, |\mathcal{F}|, |\mathcal{S}|, |\mathcal{T}|, h)$, where $|\mathcal{L}|$ is the number of couriers, $|\mathcal{R}|$ is the number of collection tasks, $|\mathcal{F}|$ is the total unoccupied capacity of the couriers, $|\mathcal{S}|$ is the location distribution of couriers and collection tasks, and $|\mathcal{T}|$ is the available time of couriers and collection tasks. The state feature vector s_h fully describes the quantitative relationship between couriers and collection tasks in the environment and their constraints on capacity, time, and spatial distribution. Second, we feed the environment state s_h into the planning module to fit an expected number of time slices (i.e., the action value a_h) based on the learned sliding window division network $\hat{Q}$. If the number of time slices currently accumulated is equal to a_h, we perform the sliding window division action. After performing the division action, the accumulated time slices will be counted from zero again. For example, as shown in Fig. 6, since the action value a_h is equal to the current accumulated time slice number h, we invoke the TBM algorithm with the inputs of the collection tasks accumulated in the last h time slices and the currently available couriers to conduct task assignments. Third, we encode decision experience as a four-entry tuple $\varphi_h = \{s_h, a_h, r_h, s_{h+1}\}$ and store it in the replay memory, where s_h is the environment state at time slice h, a_h is the number of time slices to wait, r_h is the reward (i.e., total revenue) of the matching plan returned by the TBM algorithm in the second step, and s_{h+1} the environment state at the next time slice $h+1$. For the learning module, we periodically randomly sample a batch of decision experiences $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ from the replay memory and use them to update the Q-network Q. Meanwhile, we synchronize the weight parameters of the Q-network Q to the sliding window division network $\hat{Q}$ of the planning module. Finally, as shown in Fig. 6, the above mentioned three steps are performed iteratively until a better predictive sliding window division model is learned.

Discussion. Compared with TBM algorithm, the DRL-based algorithm can determine the sliding window size reasonably to achieve a better result in quality based on the spatial-temporal distribution and supple-demand relationship between couriers and collection tasks. Although we

need a longer time to train the decision model offline, it is worth that we can get a high-quality result on short notice. In addition, the work [26] studied a fundamental bipartite graph matching problem, namely DBGM, where each node in the bipartite graph has a duration and arrives dynamically. They simplified the DBGM problem by considering only the time constraints and assumed that in a bipartite graph each node on one side can match at most one node on the other side. However, in many application scenarios of bipartite graph matching, the constraints of location and capacity are very important and cannot be ignored in real-world scenarios (e.g., carpooling, food delivery, logistics). Although our RTDW problem is also a dynamic bipartite graph matching problem, in contrast to [26], it needs to simultaneously consider many practical factors such as courier capacity, parcel location, and deadlines for delivery or collection, making the RTDW problem more complicated.

Since the action and state space of the RTDW problem is more complex than the DBGM problem, we use DQN to solve the RTDW problem rather than Q-learning adopted in [26] in order to avoid huge computation and storage costs. Furthermore, unlike [26], which only considers the number of two-side nodes in the bipartite graph to represent the environmental state, in our work we comprehensively consider five types of state features, i.e., the number of couriers, the number of collection tasks, the capacity, the location distribution, and the task completion time, which can well describe the environment and thus make the sliding window division more precise. Our experiments indeed show that the performance of our DRL algorithm is superior to that of RQL algorithm proposed in [26] in terms of average elapsed time, average revenue, and average completion ratio.

3.6 Theoretical Analysis

In this section, we analyze the competition ratios of our proposed algorithms. The competition ratio is an important indicator used to measure the performance of online algorithms [27]. The competition ratio $\mathcal{R}_c$ for our problem can be expressed as the minimum ratio between the revenue of the matching $\mathcal{M}$ produced by our online algorithms and the revenue of the offline optimal matching $\mathcal{M}_{opt}$ of the RTDW problem $\mathcal{P}$, over any input data G, S, C, and Λ.

$$\mathcal{R}_c = \min_{G,S,C,\Lambda} \frac{\mathbb{E}(\mathcal{M})}{\mathbb{E}(\mathcal{M}_{opt})} \tag{6}$$

Assumptions. We assume that the upper bound of the duration of collection tasks in an RTDW problem is $\mathcal{D}_{ub} \in \mathbb{N}$, where the existence of $\mathcal{D}_{ub}$ is reasonable since in practice the deadline of collection tasks is limited. In addition, we assume that at most one courier and one collection task appear in the same time slice and that the capacity of a courier is abundant enough. We adopt the remaining model [26] to handle the task stream because it is in line with the real-world scenario, *i.e.*, the unmatched collection tasks in a sliding window will enter the next sliding window for matching until their deadlines are reached. Next, we analyze the upper and lower bounds of the competition ratio $\mathcal{R}_c$ in Theorem 3. Note that the work [26] assumes that at most one node appears in the same time slice, but in our RTDW problem, there may be one courier and one collection task appearing in the same time slice. Therefore, the assumption of Theorem 2 in [26] is unsuitable for our problem. Since the proof method of Theorem 3 is similar to that of Theorem 2 in [26], we follow their method to prove the competitive ratio bound for the RTDW problem.

Theorem 3 (Bounds for competitive ratio $\mathcal{R}_c$). *Given an RTDW problem $\mathcal{P}$ with the duration upper bound $\mathcal{D}_{ub}$, if $\mathcal{D}_{ub} = 1$, $\mathcal{R}_c = 1$; Otherwise, if $\mathcal{D}_{ub} \geq 2$,*

$$\frac{1}{\mathcal{D}_{ub}} \leq \mathcal{R}_c < \frac{2}{\mathcal{D}_{ub} - 1} \tag{7}$$

Proof. We omit the proof due to the space limitation and put it in Appendix A.3, available in the online supplemental material. □

4 Experiments

In this section, we experimentally evaluate the efficiency and effectiveness of our proposed algorithms. We first introduce the experimental settings and then report the experimental results on both real-world datasets.

4.1 Experimental Settings

Datasets. We evaluate the algorithms over two road networks of Chengdu and New York City (NYC) extracted from OpenStreetMap.[1] Since there is no direct parcel delivery and collection task information, we extract the pick-up and drop-off points and time from two real taxi trajectory datasets collected by NYCTaxi[2] and DiDi,[3] and treat them as the parcels' collection and delivery locations and times. In practice, these pick-up and drop-off points are usually residential and working places which are also the collection and delivery points for parcels. The Chengdu dataset contains 209,423 orders, 36,630 nodes, and 50,786 edges. The NYC dataset contains 10,906,858 orders, 264,346 nodes, and 366,923 edges. Note that we divide each road network into 100 regions. The center location of each region has a station, which is responsible for parcel collection and delivery in that region. For DQN, we adopt the RMSprop algorithm as the optimizer with the learning rate of 0.001, the discount factor is set as 0.9.

Compared methods. We evaluate the performance of our proposed algorithms against two state-of-the-art methods, *RQL* [26] and *FST* [15]:

- *RQL* [26]: a Q-learning based method that adaptively decides the batch size for parcel assignment;
- *FST* [15]: a batch processing method which splits the whole parcel stream into a set of equal-size batches for matching;
- *SMA:* our greedy sequential matching algorithm described in Section 3.3;
- *TBM:* our time-aware batch matching algorithm described in Section 3.4;
- *DRL:* our DRL-based optimization described in Section 3.5;

1. https://www.openstreetmap.org
2. http://www.nyc.gov
3. http://gaia.didichuxing.com

TABLE 2
Experimental Settings

Parameters	Values
the maximum capacity, k_c	5, 15, 25, **35**, 45
# of couriers (Chengdu)	10, 30, **50**, 80, 100
# of collection tasks (Chengdu)	500, 800, **1 K**, 2 K, 3 K
# of couriers (NYC)	200, 600, **1 K**, 2 K, 3 K
# of collection tasks (NYC)	5 K, 8 K, **10 K**, 20 K, 30 K
the sliding window size, f_s	10, **20**, 30, 40, 50

Evaluation metrics and parameters. The performance of these matching algorithms is evaluated using three metrics: average total revenue, average elapsed time, and average completion ratio. The experimental parameters are summarized in Table 2, where the bold values represent the default parameter values. The algorithms are implemented in Python and tested on a machine with Intel i7-9700K @ 3.6GHz CPU and 16GB RAM.

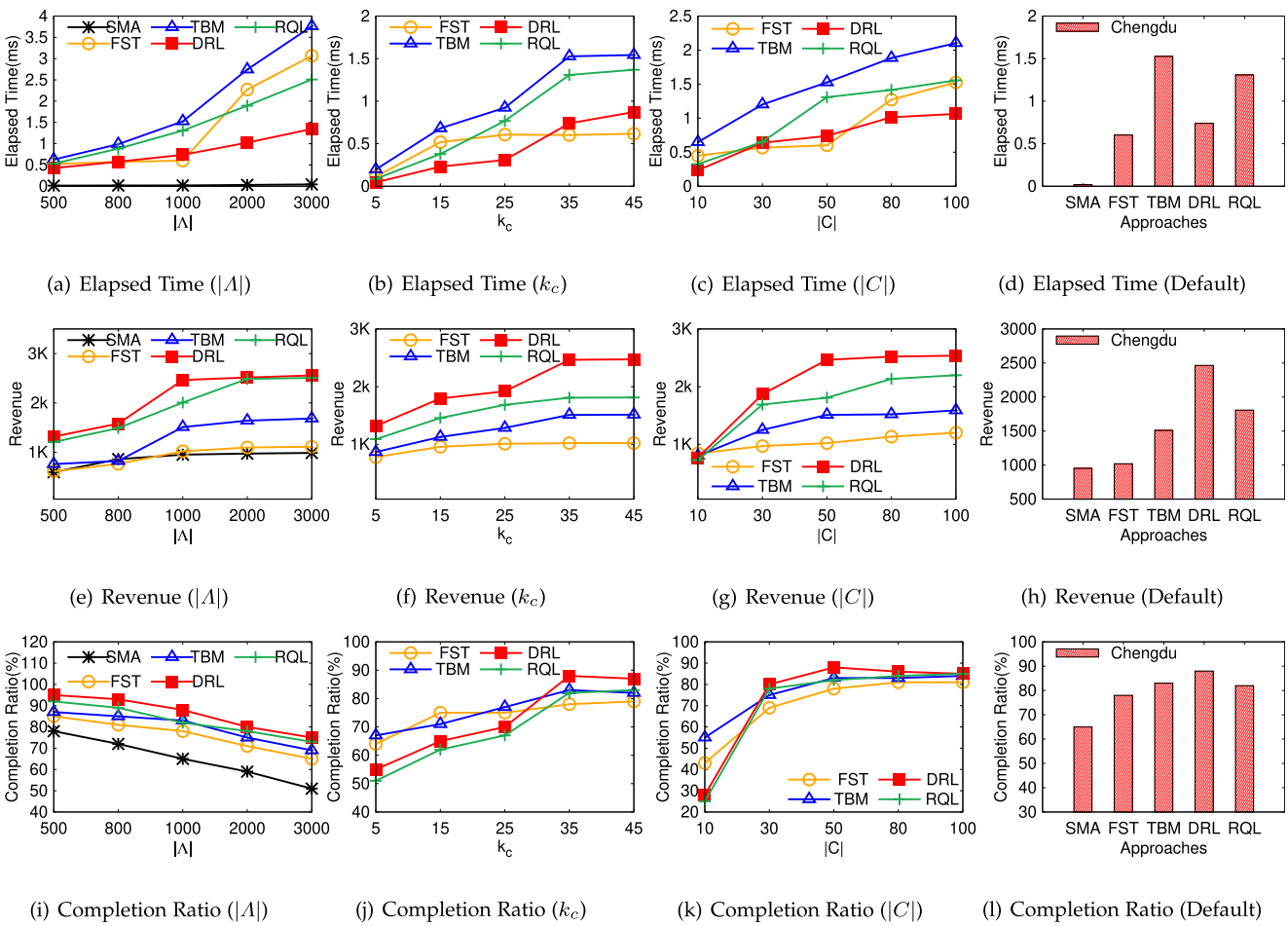
4.2 Experimental Results on Chengdu Dataset

Exp-1: Effect of sliding window with fixed size f_s. Since FST is implemented by a sliding window with a fixed size, we select an optimal sliding window size f_s=20 with the best performance (varying f_s from 10 to 50) when comparing FST with other algorithms. Table 3 shows the results of varying the size of the fixed sliding window f_s from 10 to 50. As f_s grows, the elapsed time increases, but the revenue first increases and then decreases. The reason for this is that the larger f_s is, the more collection tasks are contained in each sliding window, so the elapsed time keeps increasing. The couriers can select higher revenue collection tasks, so the revenue increases at first. However, due to the limited deadline of collection tasks, some collection tasks have expired before they are matched, which leads to a decrease in revenue and the completion ratio. When we adjust f_s from 20 to 30, the elapsed time increases by about 51%, and the revenue only increases by about 14%. Therefore, in the default setting of FST, we set $f_s = 20$.

TABLE 3
Effect of Sliding Window Size

f_s	10	20	30	40	50
Elapsed Time(ms)	0.39	**0.61**	0.92	1.38	1.59
Revenue	625	**1,020**	1,171	899	830
Completion Ratio(%)	73	**78**	83	74	70

Exp-2: Effect of the number of collection tasks $|\Lambda|$. In this set of experiments, we evaluate the compared algorithms by varying the number of collection tasks $|\Lambda|$. In Fig. 7e, as expected, the revenues of all the algorithms grow when $|\Lambda|$ is increased from 500 to 1,000. Among all the algorithms, our proposed DRL achieves the most revenue, follow by RQL, TBM, FST, and SMA. Due to the limited number of



(a) Elapsed Time ($|\Lambda|$) (b) Elapsed Time (k_c) (c) Elapsed Time ($|C|$) (d) Elapsed Time (Default)

(e) Revenue ($|\Lambda|$) (f) Revenue (k_c) (g) Revenue ($|C|$) (h) Revenue (Default)

(i) Completion Ratio ($|\Lambda|$) (j) Completion Ratio (k_c) (k) Completion Ratio ($|C|$) (l) Completion Ratio (Default)

Fig. 7. Results on Chengdu dataset.

couriers, the revenues of all the algorithms increase a little when varying $|\Lambda|$ from 1,000 to 3,000. This explains why the completion ratios of all the algorithms decrease when $|\Lambda|$ increases, as shown in Fig. 7i. In Fig. 7a, the increase of $|\Lambda|$ causes the increase of elapsed time in all the algorithms, because it needs more time to verify the valid courier-task pairs and select the best collection task for each courier. Among all the algorithms, DRL achieves the best results on revenue and completion ratio due to the adaptive sliding window. Note that we do not report the evaluation results of the SMA in the following, since it has the worst performance in terms of revenue and completion ratio.

Exp-3: Effect of the maximum capacity k_c. Figs. 7b, 7f, and 7j show the algorithms' performance when varying the maximum capacity k_c. In Fig. 7b, we can observe that the elapsed time of all the algorithms keeps increasing as k_c grows. Since the larger maximum capacity enables more couriers to become candidates for collection tasks, the searching space is enlarged. It needs more time to find the optimal solution. However, the elapsed time of FST increases from 5 to 15, almost unchanged from 15 to 45, because the completion ratio of FST in Fig. 7j almost reaches the upper bound under the fixed sliding window size. In Fig. 7f, the revenues of all the algorithms increase when the maximum capacity is enlarged. Again, DRL performs the best, followed by TBM and FST.

Exp-4: Effect of the number of couriers $|C|$. In Fig. 7c, we can see that the elapsed time of all the algorithms increases when $|C|$ grows, because they need more time to find the best courier for each collection task from among a large number of couriers. In Figs. 7g and 7k, all the algorithms achieve increasing revenues when varying the number of couriers from 10 to 50. However, when the number of couriers exceeds 50, the completion ratio and revenue of all the algorithms show little increase, since at this point most of the collection tasks are completed. In addition, from Fig. 7k, we can see that the completion ratio of DRL surpasses that of FST and TBM when the number of couriers reaches 30. However, DRL performs worse than FST and TBM when the number of couriers is 10. This is because when the number of couriers is small, the couriers cannot serve too many collection tasks in a short time due to the limited capacity. After releasing the capacity occupied by the delivery tasks, the couriers can serve more collection tasks. Simply put, it is a strategy that uses time to trade for the capacity of serving more collection tasks. In this case, our DRL approach tends to generate a larger sliding window. However, a larger sliding window delays the task assignments and may cause some collection tasks to expire, reducing the completion ratio.

Exp-5: Results on default parameter values. In this set of experiments, we study the performance of all the algorithms under default parameters. Not surprisingly, as can be seen in Figs. 7d, 7h, and 7l, compared with other approaches, the elapsed time cost of SMA is the least, owing to the face that it adopts the sequential matching strategy and only serves one collection task at a time. Accordingly, it leads to a lower completion ratio and incurs less revenue. In addition, RQL runs slower than DRL but faster than TBM since RQL's selection strategy requires checking a large Q-value table. For the revenue and completion ratio, DRL performs the best due to the use of state representation to model the environment.

4.3 Experimental Results on NYC Dataset

Exp-6: Effect of the number of collection tasks $|\Lambda|$. Figs. 8a, 8e, and 8i show the results of varying the number of requested tasks $|\Lambda|$ from 5K to 30K. When $|\Lambda|$ increases, the elapsed time of all the algorithms increases. DRL is still the fastest, followed by RQL, FST, and TBM. The results on revenue and completion ratio are similar to those for the Chengdu dataset.

Exp-7: Effect of the maximum capacity k_c. Figs. 8b, 8f, and 8j present the results for different maximum capacities of couriers k_c, from 5 to 45, while the other parameters are configured to the default values in Table 2. In Fig. 8b, the elapsed time of all the algorithms first increases and then remains unchanged when k_c increases. DRL still runs the fastest among all the algorithms and TBM runs a little slower than FST (about 9 seconds for each sliding window). In Fig. 8f, we show the revenues of all the algorithms when the maximum capacity of couriers increases. Similarly, as k_c grows, the revenue of all the algorithms increases at the beginning and then remains unchanged. DRL performs better than RQL, FST, and TBM. As shown in Fig. 8j, the completion ratio of the four approaches are close, DRL being slightly higher than RQL, FST, and TBM.

Exp-8: Effect of the number of couriers $|C|$. The results of varying the number of couriers are illustrated in Figs. 8c, 8g, and 8k. In terms of revenue as shown in Fig. 8g, DRL outperforms the other algorithms and the interval between the four is large. As regarding elapsed time, as shown in Fig. 8c, DRL is still competitive since it is faster than RQL, FST, and TBM. In Fig. 8k, at the beginning, the completion ratio of DRL is lower than that of RQL, FST, and TBM, but when $|C|$ exceeds 600, the completion ratio of DRL is slightly higher than that of RQL, FST, and TBM.

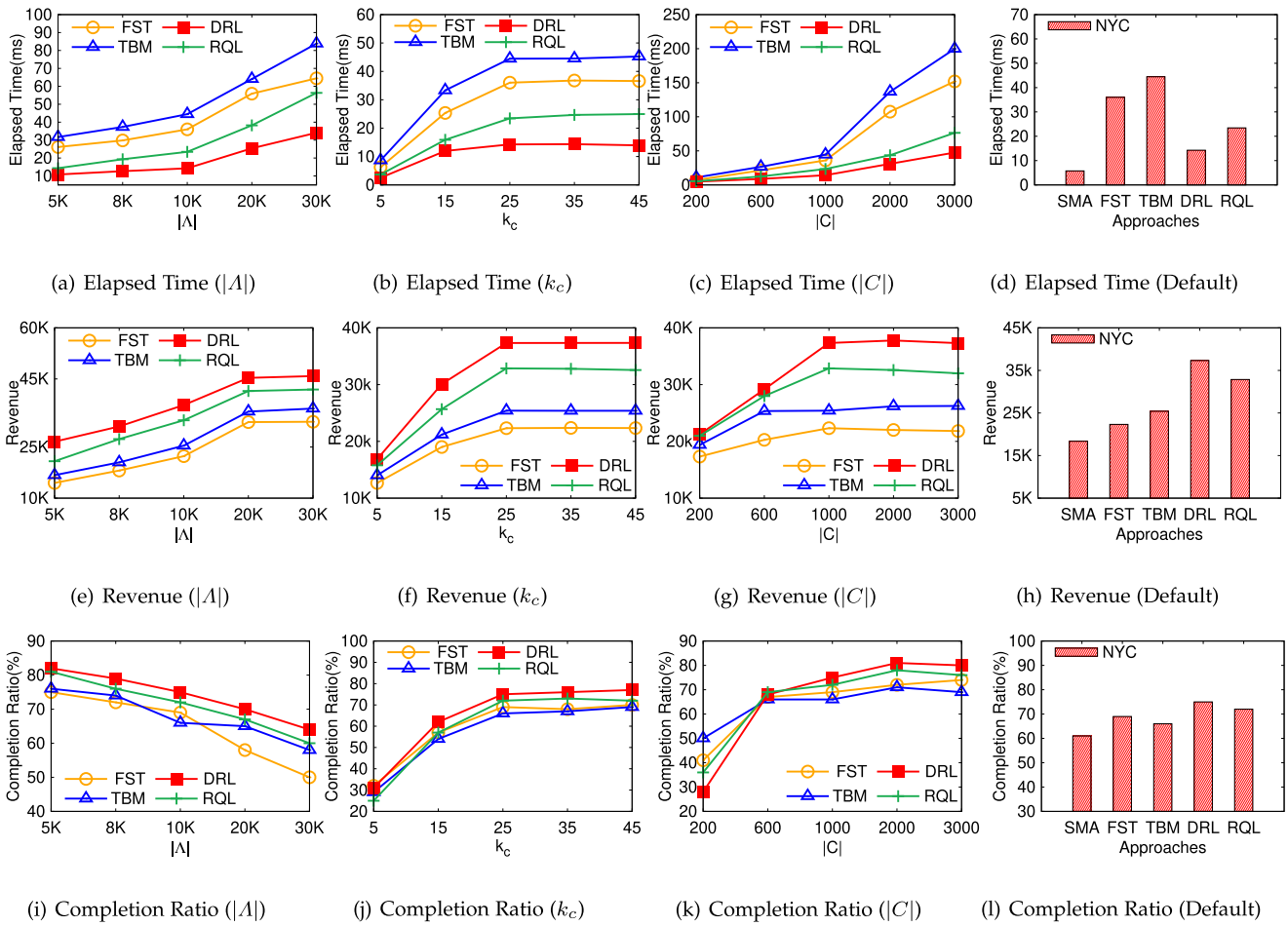
Exp-9: Results on default parameter values. Figs. 8d, 8h, and 8l report the performance of all the algorithms under the default parameter values. The performance achieved by varying parameters on the NYC dataset is similar to that on the Chengdu dataset. Figs. 8d, 8h, and 8l show the comparison results of the algorithms SMA, FST, TBM, DRL, and RQL. In particular, in Fig. 8d, we observe that DRL still runs faster than RQL. In Figs. 8h and 8l, DRL still performs the best in terms of revenue and completion ratio, followed by RQL, TBM, FST, and SMA.

Exp-10: Results on memory cost. In the last set of experiments, we examine the memory cost of the algorithms on the Chengdu and NYC datasets. As shown in Table 4, the memory costs of DRL and RQL are the largest, followed by TBM, FST, and SMA. The reason for this is that the learning based algorithms DRL and RQL always need extra space to store the learned models, *i.e.*, the DRL and RQL algorithms achieve better performance through extra memory and offline computation costs.

5 Related Work

In this section, we review three categories of related studies: city express delivery, crowdsourcing, and reinforcement learning.

City Express Delivery. In recent years, there has been a line of works studying the city express delivery problem under different settings [14], [15], [28], [29]. Zhang *et al.* [29] study



(a) Elapsed Time ($|\Lambda|$) (b) Elapsed Time (k_c) (c) Elapsed Time ($|C|$) (d) Elapsed Time (Default)

(e) Revenue ($|\Lambda|$) (f) Revenue (k_c) (g) Revenue ($|C|$) (h) Revenue (Default)

(i) Completion Ratio ($|\Lambda|$) (j) Completion Ratio (k_c) (k) Completion Ratio ($|C|$) (l) Completion Ratio (Default)

Fig. 8. Results on NYC Dataset.

a large-scale dynamic city express problem, and propose a batch processing strategy to assign collection tasks requested by customers to online couriers. In [14], Li *et al.* present a contextual collaboration reinforcement learning framework to learn courier dispatching policies and guide couriers to deliver and collect parcels for long-term revenue. In addition, several recent works have adopted a crowdsourcing approach to solve parcel delivery problems. Chen *et al.* [30] propose a taxi and passenger relay model to achieve parcel delivery, they design a two-phase framework to solve the path planning of package delivery. These studies process the parcel assignment either in a first-come-first-served mode or a fixed batch model, they do not consider optimizing parcel assignment using an adaptive approach.

Crowdsourcing. Traditional crowdsourcing that assigns tasks to suitable workers and allow workers to accomplish tasks online without moving to the locations of tasks, has attracted extensive attention from various fields [31], [32]. It has many successful high practical applications in real-world, such as Upwork and Amazon Mechanical Turk. Recently, spatial crowdsourcing has gradually replaced traditional crowdsourcing with the rapid development of wireless networks and sharing economy. Compared with traditional crowdsourcing, spatial crowdsourcing, which requires workers physically traveling to locations of spatial tasks, has stronger spatio-temporal constraints. It has many application scenarios, such as ridesharing [20], [33], [34], query [35], [36], city express [29], food delivery [37]. In this paper, the RTDW problem is one of the spatial crowdsourcing problems that aim to solve the assignment problem through adaptive sliding window division.

Reinforcement Learning. Reinforcement learning has been extensively studied in the fields of resource allocation [14], [38], [39], decision-making [26], [40], [41], and robot control [42], [43]. Shan *et al.* [39] propose a DRL framework for task scheduling in crowdsourcing, they utilize an RL-based method to estimate the expected long-term revenue of task recommendation. Zhou *et al.* [44] propose a decentralized execution order-dispatching method based on multi-agent reinforcement learning to solve a large-scale order dispatch problem. In [45], Lin *et al.* study a large-scale online ride-sharing platform in a dynamic supply and demand environment, they propose two reinforcement learning based algorithms to solve a large-scale fleet management problem such that a large number of agents can efficiently fit different contexts. Zhang *et al.* [40] design an end-to-end

TABLE 4
Results on Memory Cost

Datasets(MB)	SMA	FST	TBM	DRL	RQL
Chengdu	15.3	21.6	24.7	61.7	73.8
NYC	132.4	175.3	189.3	492.7	565.2

DRL network to find the best configuration in a high-dimensional continuous database management system space. To the best of our knowledge, we are the first to study the RTDW problem, and the methods described above cannot be directly applied to solve our problem.

6 Conclusion and Future Work

In this paper, we study the problem of real-time city express delivery and prove its hardness. To address this problem, we present an efficient TBM algorithm to perform matching within a sliding window. We also propose a DRL-based optimization to further enhance the solution quality. Extensive experimental results confirm the effectiveness and efficiency of our proposed algorithms.

As for future work, we plan to work on the following two extensions: One is how to reasonably layout express stations to further improve the efficiency of express delivery services; the other is how to develop an efficient collaborative model to facilitate courier cooperation such that parcels can be delivered and collected by a group of couriers.

References

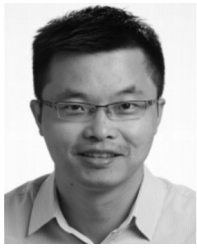
[1] China express industry development report (2019–2020), 2020. [Online]. Available: http://www.spbdrc.org.cn/
[2] Amazon, 2021. [Online]. Available: https://www.amazon.com/
[3] Jd, 2021. [Online]. Available: https://www.jd.com/
[4] Cainiao, 2021. [Online]. Available: https://www.cainiao.com/
[5] Y. Zhao, K. Zheng, Y. Cui, H. Su, F. Zhu, and X. Zhou, "Predictive task assignment in spatial crowdsourcing: A data-driven approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 13–24.
[6] J. Xia, Y. Zhao, G. Liu, J. Xu, M. Zhang, and K. Zheng, "Profit-driven task assignment in spatial crowdsourcing," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 1914–1920.
[7] Y. Zhao *et al.*, "Preference-aware task assignment in spatial crowdsourcing," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019, pp. 2629–2636.
[8] T. Song *et al.*, "Trichromatic online matching in real-time spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2017, pp. 1009–1020.
[9] P. Cheng, L. Chen, and J. Ye, "Cooperation-aware task assignment in spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1442–1453.
[10] W. Ni, P. Cheng, L. Chen, and X. Lin, "Task allocation in dependency-aware spatial crowdsourcing," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 985–996.
[11] B. Li, Y. Cheng, Y. Yuan, G. Wang, and L. Chen, "Simultaneous arrival matching for new spatial crowdsourcing platforms," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2020, pp. 1279–1287.
[12] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: A survey," *Proc. VLDB Endowment*, vol. 29, no. 1, pp. 217–250, 2020.
[13] M. Li *et al.*, "Privacy-preserving batch-based task assignment in spatial crowdsourcing with untrusted server," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, 2021, pp. 947–956.
[14] Y. Li and Y. Zheng and Q. Yang, "Efficient and effective express via contextual cooperative reinforcement learning," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 510–519.
[15] Y. Li, Y. Zheng, and Q. Yang, "Cooperative multi-agent reinforcement learning in express system," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manage.*, 2020, pp. 805–814.
[16] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 7, pp. 1782–1795, Jul. 2015.
[17] P. Cheng, H. Xin, and L. Chen, "Utility-aware ridesharing on road networks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 1197–1210.
[18] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 410–421.
[19] L. Zheng, L. Chen, and J. Ye, "Order dispatch in price-aware ridesharing," *Proc. VLDB Endowment*, vol. 11, no. 8, pp. 853–865, 2018.
[20] L. Chen, Q. Zhong, X. Xiao, Y. Gao, P. Jin, and C. S. Jensen, "Price-and-time-aware dynamic ridesharing," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 1061–1072.
[21] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 773–788.
[22] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231.
[23] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi, "Geo-social k-cover group queries for collaborative spatial computing," in *Proc. IEEE Int. Conf. Data Eng.*, 2016, pp. 1510–1511.
[24] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 1994.
[25] H. Ye, D. Zhan, X. Si, and Y. Jiang, "Learning mahalanobis distance metric: Considering instance disturbance helps," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 3315–3321.
[26] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1478–1489.
[27] M. Tennenholtz, "Rational competitive analysis," in *Proc. 17th Int. Joint Conf. Artif. Intell.*, 2001, pp. 1067–1072.
[28] A. Sadilek, J. Krumm, and E. Horvitz, "Crowdphysics: Planned and opportunistic crowdsourcing for physical tasks," in *Proc. 7th Int. Conf. Weblogs Social Media*, 2013, pp. 536–545.
[29] S. Zhang, L. Qin, Y. Zheng, and H. Cheng, "Effective and efficient: Large-scale dynamic city express," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3203–3217, Dec. 2016.
[30] C. Chen *et al.*, "Crowddeliver: Planning city-wide package delivery paths leveraging the crowd of taxis," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 6, pp. 1478–1496, Jun. 2017.
[31] L. Cui, J. Chen, W. He, H. Li, W. Guo, and Z. Su, "Achieving approximate global optimization of truth inference for crowdsourcing microtasks," *Proc. Data Sci. Eng.*, vol. 6, no. 3, pp. 294–309, 2021.
[32] A. I. Chittilappilly, L. Chen, and S. Amer-Yahia , "A survey of general-purpose crowdsourcing techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 9, pp. 2246–2266, Sep. 2016.
[33] Y. Li, R. Chen, L. Chen, and J. Xu, "Towards social-aware ridesharing group query services," *IEEE Trans. Serv. Comput.*, vol. 10, no. 4, pp. 646–659, Jul./Aug. 2017.
[34] Y. Li *et al.*, "Top-k vehicle matching in social ridesharing: A price-aware approach," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 3, pp. 1251–1263, Mar. 2021.
[35] Y. Li, R. Chen, J. Xu, Q. Huang, H. Hu, and B. Choi, "Geo-social k-cover group queries for collaborative spatial computing," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 10, pp. 2729–2742, Oct. 2015.
[36] L. Chen, Y. Li, J. Xu, and C. S. Jensen, "Towards why-not spatial keyword top- queries: A direction-aware approach," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 796–809, Apr. 2018.
[37] Y. Liu *et al.*, "FooDNet: Toward an optimized food delivery network based on spatial crowdsourcing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1288–1301, Jun. 2019.
[38] S. Tian, S. Mo, L. Wang, and Z. Peng, "Deep reinforcement learning-based approach to tackle topic-aware influence maximization," *Proc. Data Sci. Eng.*, vol. 5, no. 1, pp. 1–11, 2020.
[39] C. Shan, N. Mamoulis, R. Cheng, G. Li, X. Li, and Y. Qian, "An end-to-end deep RL framework for task arrangement in crowdsourcing platforms," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 49–60.
[40] J. Zhang *et al.*, "An end-to-end automatic cloud database tuning system using deep reinforcement learning," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2019, pp. 415–432.
[41] Z. Gu, T. Yin, and Z. Ding, "Path tracking control of autonomous vehicles subject to deception attacks via a learning-based event-triggered mechanism," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 12, pp. 5644–5653, Dec. 2021.
[42] Z. Bing, C. Lemke, Z. Jiang, K. Huang, and A. Knoll, "Energy-efficient slithering gait exploration for a snake-like robot based on reinforcement learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 5663–5669.
[43] Y. Zhao, Y. Ma, and S. Hu, "USV formation and path-following control via deep reinforcement learning with random braking," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 12, pp. 5468–5478, Dec. 2021.

[44] M. Zhou *et al.*, "Multi-agent reinforcement learning for order-dispatching via order-vehicle distribution matching," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, 2019, pp. 2645–2653.
[45] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 1774–1783.
[46] F. Focacci, A. Lodi, and M. Milano, "A hybrid exact algorithm for the TSPTW," *INFORMS J. Comput.*, vol. 14, no. 4, pp. 403–417, 2002.

Yafei Li received the PhD degree in computer science from Hong Kong Baptist University, in 2015. He is currently an associate professor with the School of Computer and Artificial Intelligence, Zhengzhou University, China. His research interests include span mobile and spatial data management, location-based services, and urban computing. He has authored more than 20 journal and conference papers in these areas, including *IEEE Transactions on Knowledge and Data Engineering*,*IEEE Transactions on Services Computing*, *Transactions on the Web*, *ACM Transactions on Intelligent Systems and Technology*, *Proceedings of the VLDB Endowment*, IEEE ICDE, WWW, etc.

Qingshun Wu received the BEng degree in computer science and technology from Zhengzhou University, China, in 2019. He is currently working toward the MEng degree with the School of Computer and Artificial Intelligence, Zhengzhou University. His research interests include multi-agent computing, deep learning, and spatiotemporal data processing.

Xin Huang received the PhD degree from the Chinese University of Hong Kong (CUHK) in 2014. He is currently an assistant professor with Hong Kong Baptist University. His research interests mainly focus on graph data management and mining.



Jianliang Xu received the BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, and the PhD degree in computer science from the Hong Kong University of Science and Technology. He is currently a professor with the Department of Computer Science, Hong Kong Baptist University. He held visiting positions with Pennsylvania State University and Fudan University. His research interests include Big Data management, mobile computing, and data security and privacy. He has published more than 200 technical papers in these areas. He has served as a program cochair/vice chair for a number of major international conferences including IEEE ICDCS 2012, IEEE CPSNA 2015, and APWeb-WAIM 2018. He is an associate editor for *IEEE Transactions on Knowledge and Data Engineering* and *Proceedings of the VLDB Endowment* 2018.

Wanru Gao received the PhD degree from the School of Computer Science, University of Adelaide, Australia, in 2016. She is currently a lecturer with the School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. Her current research interests include the combinatorial optimization, diversity maximization in evolutionary algorithms and theoretical analysis of heuristic search methods. She has authored more than 20 journal and conference papers in these areas.

Mingliang Xu received the PhD degree from the State Key Lab of CAD&CG, Zhejiang University, China. He is a professor with the School of Computer and Artificial Intelligence, Zhengzhou University, China. His current research interests include computer graphics, multimedia and artificial intelligence. He has authored more than 60 journal and conference papers in these areas, including *ACM Transactions on Graphics*, *IEEE Transactions on Pattern Analysis and Machine Intelligence/TIP*/*IEEE Transactions on Circuits and Systems for Video Technology*, ACM SIGGRAPH (Asia)/MM, ICCV, etc.