Effective Multi-User Delay-Constrained Scheduling with Deep Recurrent Reinforcement Learning

Pihe Hu, Ling Pan, Yu Chen, Zhixuan Fang, Longbo Huang

Tsinghua University

Beijing, China

{hph19,pl17,c-y19}@mails.tsinghua.edu.cn,{zfang,longbohuang}@tsinghua.edu.cn

ABSTRACT

Multi-user delay constrained scheduling is important in many real-world applications including wireless communication, live streaming, and cloud computing. Yet, it poses a critical challenge since the scheduler needs to make real-time decisions to guarantee the delay and resource constraints simultaneously without prior information of system dynamics, which can be time-varying and hard to estimate. Moreover, many practical scenarios suffer from partial observability issues, e.g., due to sensing noise or hidden correlation. To tackle these challenges, we propose a deep reinforcement learning (DRL) algorithm, named Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient (RSD4)[1], which is a data-driven method based on a Partially Observed Markov Decision Process (POMDP) formulation. RSD4 guarantees resource and delay constraints by Lagrangian dual and delay-sensitive queues, respectively. It also efficiently tackles partial observability with a memory mechanism enabled by the recurrent neural network (RNN) and introduces user-level decomposition and node-level merging to ensure scalability. Extensive experiments on simulated/real-world datasets demonstrate that RSD4 is robust to system dynamics and partially observable environments, and achieves superior performances over existing DRL and non-DRL-based methods.

CCS CONCEPTS

• **Networks** → **Network resources allocation**; • **Computing methodologies** → **Planning for deterministic actions**.

KEYWORDS

Delay-constrained, Scheduling, Partial Observability, Deep Reinforcement Learning

[1]Code available at https://github.com/hupihe/RSD4

1 INTRODUCTION

Due to the emergence of many real-time interactive applications, e.g., online games, virtual reality (VR), and cloud computing, and the increasingly more rigid user requirements, delay-constrained scheduling has become a central problem in guaranteeing the satisfying quality of experience in many areas. For example, express delivery is a typical delay-sensitive scheduling problem. According to [17], a small increase in the delivery time will significantly impact customers' perceived ambiguity and risk, and reduce satisfaction. Delay-constrained scheduling is also critical for data communications [34], video streaming [2], and data center management [25].

However, delay-constrained scheduling is challenging in many aspects. First, the scheduler needs to satisfy latency and resource constraints, while the delay metric depends on the overall dynamics and control of the system across time, and the resource constraint further couples scheduling decisions. Second, system dynamics, e.g., user channels in mobile networks, are hard to trace since distributions of underlying random components can be highly dynamic and correlated. Third, practical scheduling systems usually face a large number of users and have complex network structures, and demand highly scalable solutions for large-scale and multi-hop scenarios. For example, live video platforms such as YouTube and Instagram have thousands of millions of daily active users [4]. Fourth, due to sensing noise and hidden correlation, practical systems can also suffer from partial observability issues, e.g., most IoT devices cannot have perfect knowledge of a dynamic channel environment due to hardware limitation and short sensing time [32], and channel states in network systems can also be hard to obtain [14]. This universality of partial observability in real-world scheduling problems demands highly robust algorithms.

Many algorithms have been proposed for scheduling problems based on different methods, including queueing-based methods, e.g., [11, 26], optimization-based methods, e.g., [10, 13], dynamic programming (DP)-based algorithms, e.g., [3, 28], and Lyapunov control-based approaches, e.g., [16, 23]. However, these approaches either require prior knowledge about system dynamics, or suffer from curse-of-dimensionality due to large state space, or focus on stability constraints rather than delay. To overcome the above limitations, we propose a deep reinforcement learning (DRL)-based algorithm, named Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient (RSD4), whose procedure is depicted in Figure 1. RSD4 builds upon the recurrent deterministic policy gradient [8] and softmax deterministic policy gradient [22], and introduces several novel components for handling the scheduling problem in partially observed settings. Specifically, RSD4 is an end-to-end method based on a partially observed Markov decision process (POMDP) formulation with a Lagrange dual update. It does

not require any prior knowledge of system dynamics and effectively captures hidden system correlation across time slots by a recurrent module. RSD4 makes use of a softmax operator to improve value estimation in training and robustness in handling complex system dynamics. Finally, it introduces unified training for improving training efficiency, and user-level decomposition and node-level merging to support large-scale and multi-hop scenarios.
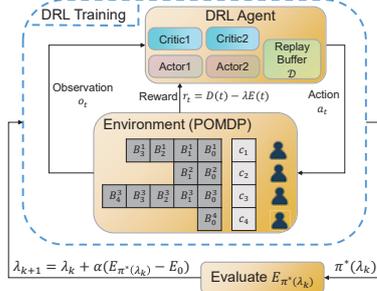

Figure 1: RSD4 framework for delay-constrained scheduling.

The RSD4 framework has several unique features that make it suitable for handling delay-sensitive scheduling tasks, especially in partially observable systems. Firstly, its base POMDP formulation is general to capture the partial observability issue that often arises in real-world systems. Secondly, the proposed RSD4 algorithm possesses key advantages of existing DRL algorithms, i.e., being a data-driven and end-to-end method that does not require prior knowledge of the system dynamics. Thirdly, the RSD4 algorithm is highly scalable, making it suitable for practical implementation in large-scale resource-constrained scenarios, where the system observation can be partial and even not instantaneous. We conduct extensive experiments to validate the performance of RSD4 on both simulated environments and real-world datasets. The results demonstrate that RSD4 significantly outperforms classical non-DRL methods and existing DRL benchmarks in various scenarios. Our framework and results shed light on designing scalable and efficient DRL algorithms for scheduling complex systems.

The main contributions of this paper are summarized as follows.

(i) We formulate a general POMDP framework for investigating large-scale multi-user delay-constrained scheduling problems with average resource constraint, which is suitable for partially observable systems, and provides two novel functions for scalability, i.e., user-level decomposition and node-level merging.

(ii) We propose a novel DRL-based algorithm, RSD4, to tackle the partial observability problem and achieve robust value prediction. RSD4 builds upon the recurrent policy gradient method and softmax-based value estimation. It introduces a unified training approach to improve training efficiency. It also adopts a double-branch neural network architecture and delayed policy update to learn a robust policy against different system dynamics.

(iii) We conduct extensive experiments on real datasets, showing that RSD4 efficiently learns system dynamics under both large-scale and multi-hop cases, and significantly outperforms existing scheduling methods, especially in various partially-observable settings.

2 RELATED WORK

Many existing works have studied the scheduling problem. Among the many techniques adopted, four methodologies have received much attention, including queueing theory-based method, e.g., [11, 26], optimization-based method, e.g., [10, 13], dynamic programming-based control, e.g., [3, 28], Lyapunov-based optimization, e.g., [16, 23]. However, these approaches either do not explicitly capture the delay constraint or require precise system dynamics which can be highly non-trivial in real systems. Besides, DP-based algorithms often suffer from the curse-of-dimensionality and are not scalable to large-scale scenarios. DRL has been receiving much attention in the scheduling field, due to its generalization and scalability capability, and has been adopted in several scheduling scenarios, e.g., video streaming [18], Multipath TCP control [34], network re-configurability [2], MAC scheduling [20], and resource-constrained scheduling, e.g., [7, 19, 21, 31]. However, the aforementioned works either do not ensure average resource constraints or require a perfectly observable system state as input. In addition, they do not consider large-scale or multi-hop systems.

Our proposed RSD4 is a data-driven end-to-end algorithm, which well handles the partial observability issue under the POMDP formulation. Besides, the delay and resource constraint are handled by delay-sensitive queues and the Lagrangian dual, respectively. It also adopts user-level decomposition and node-level merging to significantly extend the scalability.

3 PROBLEM AND PRELIMINARY

In this section, we present our scheduling problem description. For ease of presentation, we focus on the single-hop scheduling problem in Section 3.1 and the corresponding Lagrangian dual in Section 3.2. The multi-hop problem will be presented later in Section 4.3.

3.1 The Scheduling Problem

We consider the scheduling problem illustrated in Figure 2. Time is divided into discrete slots $t \in \{0, 1, 2, ...\}$. At the beginning of each time slot, the scheduler first receives job arrivals, e.g., data packets in a network or parcels in a delivery station. The number of job arrivals for user i at time slot t is denoted as $A_i(t)$. We denote $A(t) = [A_1(t), ..., A_N(t)]$. Each job for user i is associated with a strict delay constraint τ_i, i.e., a job needs to be served within τ_i slots upon its arrival and will be outdated and discarded otherwise.
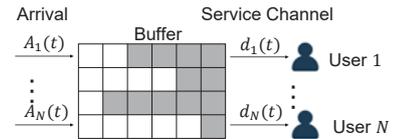

Figure 2: A general delay-constrained scheduling problem in a single-hop network. Jobs arrive at the server and need to be delivered to their destinations before deadline.

The buffer model. Jobs arriving at the system are first placed in a buffer, conveniently modeled by a set of delay-sensitive queues. Specifically, the buffer contains N separate delay-sensitive queues of infinite size, one for each user. The state of each queue i at time slot t is denoted by $B_i(t) = [B_i^0(t), B_i^1(t), ..., B_i^{\tau_i}(t)]$, where $B_i^{\tau}(t)$ stands for the number of jobs for user i with a remaining time of τ timeslots until expiration for $1 \le \tau \le \tau_i$.

The scheduling and service model. At every time slot t, the scheduler makes decision on the resources allocated to jobs in the buffer. The decision for user i is denoted as $e_i(t) = [e_i^0(t), e_i^1(t), ..., e_i^{\tau_i}(t)]$,

where $e_i^\tau(t) \in [0, e_{\max}]$ denotes the resource, e.g., energy, allocated to serve each job in queue i with deadline τ. Each scheduled job then goes through a service channel, e.g., a wireless channel, whose condition is random and its value at time slot t for user i is denoted by $c_i(t)$. We denote the service channel conditions at timeslot t as $\boldsymbol{c}(t) = [c_1(t), c_2(t), ..., c_N(t)]$. For a user i job with allocated resource e and channel condition c, the probability of successful service is denoted as $P_i(e, c)$, where $e = 0$ means that the job will not be served and $P_i(0, c) = 0$. Also, $P_i(e, \cdot) > 0$ for all $e > 0$. If a job is scheduled but fails in service, it remains in the buffer if it is not outdated. The instantaneous resource consumption at time slot t is denoted as $E(t) = \sum_{i=1}^N \boldsymbol{e}_i^\top(t) \boldsymbol{B}_i(t)$ and the average resource consumption is $\overline{E} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T E(t)$.

The system objective. For user i, the number of successfully served jobs at time slot t is denoted as $d_i(t)$. Each user is given a weight β_i, and the weighted instantaneous throughput is denoted as $D(t) = \sum_{i=1}^N \beta_i d_i(t)$. The objective of the scheduler is to maximize the weighted average throughput, defined as $\overline{D} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T D(t)$, subject to the average resource consumption limit, i.e., [2]

$$\mathcal{P} \quad \max_{\boldsymbol{e}_i(t): 1 \leq i \leq N, 1 \leq t \leq T} \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N \beta_i d_i(t) \tag{1}$$

$$\text{s.t.} \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N \boldsymbol{e}_i^\top(t) \boldsymbol{B}_i(t) \leq E_0$$

where E_0 is the average resource consumption limit. We denote its optimal value as $\mathcal{T}^*$.

Note that although our problem description adapts the formulation in [28], our goal is to develop practical and scalable solutions to scheduling in real-world environments, i.e., without assumptions on ergodicity of dynamics and allow the randomness to be arbitrarily correlated with hidden factors. Note here we allow resources to be allocated without hard constraint. We will show in Section 6.3 that hard constraints can be easily incorporated with only minimal impact on performance.

3.2 Lagrange Dual

Define the following Lagrangian function of problem (1) for handling the average resource budget E_0:

$$\mathcal{L}(\pi, \lambda) = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N [\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t) \boldsymbol{B}_i^\top(t)] + \lambda E_0 \tag{2}$$

with respect to control policy π and Lagrange multiplier λ. Denote $g(\lambda)$ the Lagrange dual function for fixed Lagrange multiplier λ:

$$g(\lambda) = \max_\pi \mathcal{L}(\pi, \lambda) = \mathcal{L}(\pi^*(\lambda), \lambda), \tag{3}$$

where the maximizer is denoted as $\pi^*(\lambda)$. Using Lemma 3 in [28], the optimal timely throughput $\mathcal{T}^*$ equals the optimal value of the dual problem, i.e., $\mathcal{T}^* = \min_{\lambda \geq 0} g(\lambda) = g(\lambda^*)$, where λ^* is the optimal Lagrange multiplier. To find the optimal policy $\pi^*(\lambda^*) = \arg\max_\pi \mathcal{L}(\pi, \lambda^*)$, it suffices to find optimal Lagrangian multiplier λ^*. Denote the consumed resource under policy π in time slot t as $E_\pi(t)$. The Danskin's Theorem in [1] states that $g'(\lambda) = \frac{\partial \mathcal{L}(\pi^*, \lambda)}{\partial \lambda} =$

$E_0 - E_{\pi^*(\lambda)}$, where $E_{\pi^*(\lambda)} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T E_{\pi^*(\lambda)}(t)$ denotes the average resource consumption. Thus, the optimal policy $\pi^*(\lambda^*)$ can be obtained by recovering the dual function $g(\lambda)$ for some λ and taking gradient descent to find the optimal λ^*. While the theoretical understanding is clear, finding the optimal $\pi^*(\lambda)$ for a given λ in a practical system is non-trivial, due to the following main reasons:

- System dynamics are hard to trace since distributions can be highly dynamic and correlated in many scenarios.
- The system can only be partially observed, e.g., due to sensing limitation and noise and other hidden factors.
- Practical scheduling systems usually have a large number of users and complex multi-hop topology, which demand highly scalable solutions.

Prior works [3, 28] uses DP to find the maximizer $\pi^*(\lambda)$, which requires the prior knowledge of system dynamics and suffers from the curse-of-dimensionality in large systems, and may not directly apply to partially observable systems. These motivate us to design a DRL-based framework with POMDP formulation in Section 4.

4 OVERALL FRAMEWORK

In this section, we present our novel POMDP formulation for solving the scheduling problem $\mathcal{P}$ in Section 3.1, which supports user-level decomposition and node-level merging for scalability.

4.1 The POMDP Formulation

We now specify the POMDP formulation for the scheduling problem, under which the RSD4 can be applied to find the optimal policy $\pi^*(\lambda^*)$. Specifically, the POMDP is represented by $\mathcal{M} = \langle \mathcal{S}, O, \mathcal{A}, r, P, \gamma \rangle$, where $\mathcal{S}$ is the state space, O is the observation space, $\mathcal{A}$ is the action space, r is the reward function, P denotes the transition matrix, and γ stands for the discount factor.

State and Observation. The overall system state s_t includes $\boldsymbol{A}(t)$, $\boldsymbol{B}_1(t), ..., \boldsymbol{B}_N(t)$, $\boldsymbol{c}(t)$ and other information of the underlying MDP unobservable by the scheduler, e.g., random hyperparameters of successful transmission probability, or mobile user position that affects traffic arrival and channel. We consider s_t to be partially-observed since partial observable scenarios are common in scheduling problems. This implies that the actual observation o_t is a subset of s_t and the exact form of o_t depends on environment settings. For example, Figure 3(a) shows the observed system of a four-use case, where the observation $o_t = [\boldsymbol{B}_1(t), ..., \boldsymbol{B}_N(t), \boldsymbol{c}(t)]$ only includes buffer with delay-sensitive queues and channel states but not other factors.

(a) Observed system   (b) Decomposed observation
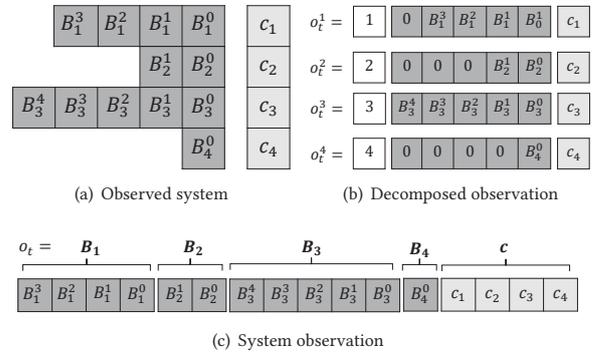

(c) System observation

Figure 3: Observations for a four-user system (time index omitted for brevity).

Action and Reward. At time slot t, the control action is denoted by $a_t = [e_1(t), ..., e_N(t)]$, and the reward is set to

$$r_t = D(t) - \lambda E(t), \qquad (4)$$

which is the instantaneous weighted throughput $D(t)$ minus the resource consumption $E(t)$ weighted by the multiplier λ.

Learning Objective. Under POMDP, an optimal agent needs to access the entire history $h_t = (o_1, a_1, o_2, a_2, ..., a_{t-1}, o_t)$ and learn a deterministic policy $\pi(\cdot; \phi)$ parameterized by ϕ, which maps from the history to the action space, with the objective of maximizing the expected long-term rewards

$$J(\pi(\cdot; \phi)) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \mid s_0, a_0, \pi(\cdot; \phi) \right].$$

REMARK 1. *With the reward setting in (4), when $\gamma = 1$, the cumulative discounted reward is $R = \sum_{t=1}^{T} \gamma^t r_t = T\mathcal{L}(\pi, \lambda) - \lambda T E_0$, which differs from the Lagrangian function value in (2) by $-\lambda T E_0$. This means that an algorithm maximizing the expected rewards $J(\pi(\cdot; \phi))$ is also a maximizer for the Lagrangian function, which is the objective of* RSD4 *(detailed in Section 5).*

4.2 User-Level Decomposition

The system state contains the buffer information whose size is proportional to the user number. This makes learning much harder in large-scale systems, due to the need to train neural networks with more hyperparameters, which largely limits the scalability of DRL-based methods. To overcome this issue, we propose a *user-level decomposition* technique, such that RSD4 finds the optimal policy with small neural networks even under large-scale scenarios. Our user-level decomposition is different from the packet-level decomposition in [3, 28], primarily due to the fact that packet-level decomposition requires a much larger number of neural network parameters, resulting in poor performance.

Specifically, we define the user-level decomposed Lagrangian function for user i as

$$\mathcal{L}_i(\pi_i, \lambda_i) = \lim_{T \to \infty} \frac{1}{T} \left[\sum_{t=1}^{T} \beta_i d_i(t) - \lambda_i e_i^\top(t) B_i^\top(t) \right], \qquad (5)$$

where π_i is the decomposed policy for scheduling user i's jobs in the buffer. Consequently, maximizing the Lagrangian function in (2) for a fixed λ can be accomplished by maximizing (5) for each user i's sub-problem separately with $\lambda_i = \lambda$. Denote the optimal policy for user i's sub-problem as $\pi_i^*(\lambda_i)$ with multiplier λ_i. After solving Problem (5) for each user with a fixed λ_i, the optimal policy $\pi^*(\lambda)$ that maximizes the Lagrangian (2) can be derived by letting each user take its own optimal scheduling policy $\pi_i^*(\lambda_i)$ with $\lambda_i = \lambda$.

Decomposed POMDP. Based on the above intuition, we decompose the POMDP into user-level subproblems, where for each user i, the action is $a_t^{(i)} = e_i(t)$, and the reward becomes $r_t^{(i)} = \beta_i d_i(t) - \lambda e_i^\top(t) B_i(t)$ at time slot t. The observation is also decomposed as well, Figure 3(b) presents $o_t^{(i)} = [i, B_i(t), c_i(t)]$ for a four-user case, where the first index for user i's sub-problem is required for algorithm training (explained next). This is a key step in RSD4. As we will see in Section 6.5, without decomposition, RSD4 and other existing DRL algorithms can fail due to a large size of the state representation.

Unified Training. After decomposition, the state space is compressed, and there are N different sub-POMDPs, whose dynamics of different users may not be the same. To avoid training N different DRL agents, leading to linear growth of computation power, we propose the method of unified training. That is, to train different samples from different sub-environments together, and use an extra user index i as the identifier to samples from the user i's sub-problem, as shown in Figure 3(b). Consequently, the dimension of training samples remains the same regardless of the system scale.

REMARK 2. *With observation decomposition and unified training, a single system observation o_t is decomposed into N separate sate $o_t^1, o_t^2, ..., o_t^N$, such that one step dynamics in the original environment creates N samples for the replay buffer. Consequently, it does not require heavy parallel computation and greatly enriches the abundance of the replay buffer, such that common knowledge across different users' sub-POMDPs is learned efficiently. With PODMP decomposition, the number of neural network parameters also remains small even for large-scale systems, because the dimension of observation remains unchanged after decomposition. This useful feature makes our framework highly scalable.*

4.3 Multi-Hop and Node-level Merging

We present the multi-hop setting here and describe a technique called *node-level merging* for reducing complexity in this case. Specifically, in multi-hop networks, each flow can traverse multiple hops and the agent needs to decide which flows to serve and how many resources to allocate at each node. Besides, each node has an average resource constraint. Take Figure 4(a) as an example. There are three flows passing through three paths respectively, i.e., $F_1 = \{1 \to 2 \to 3 \to 5\}$ with deadline τ_1, $F_2 = \{2 \to 4 \to 6\}$ with deadline τ_2, and $F_3 = \{2 \to 3\}$ with deadline τ_3. The idea of

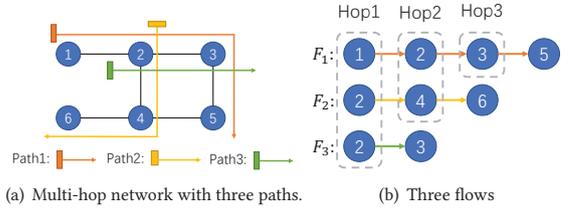(a) Multi-hop network with three paths. (b) Three flows

Figure 4: (a) A multi-hop network with three paths represented in different colors, each supports one flow. (b) The flows are aligned with starting nodes.

node-level merging is to augment the state, observation, action, and reward in the POMDP formulation for multihop scheduling. Figure 5 shows one example with the detailed procedure below.

- *State and Observation*: The system state is obtained by merging buffer and channel states in different hops, as shown in Figure 5. Since jobs in a multi-hop flow differ only in node position and remaining time until expiration, we encode the system state by an aggregation of buffer states at the nodes. Besides, the buffer state at hop j for flow i is denoted as $B_i^{(j)}(t) = [B_i^{0(j)}(t), B_i^{2(j)}(t), ..., B_i^{\tau_i(j)}(t)]$ where $B_i^{\tau(j)}(t)$ denotes the number of jobs for flow i at hop j with a remaining time of τ timeslots until expiration for $0 \le \tau \le \tau_i$. Denote the path length flow i as h_i. The node-level merging idea is to set the overall observation o_t to

be $[B_1^{(1)}(t), ..., B_1^{(h_1)}(t), ..., B_N^{(h_N)}(t), c(t)]$ and potentially other factors in the environment. Figure 5 shows a case when $o_t = [B_1^{(1)}(t), ..., B_1^{(h_1)}(t), ..., B_N^{(h_N)}(t), c(t)]$.

- *Action*: The action is similarly obtained by merging actions at different nodes, i.e., $a_t = [e_1^{(1)}(t), ..., e_1^{(h_1)}(t), ..., e_N^{(h_N)}(t)]$, where $e_i^{(j)}(t)$ represent the resource allocated to user i's j-th node for $1 \leq j \leq h_i$.
- *Reward*: The reward is set to $r_t = D(t) - \sum_{k=1}^{M} \lambda_k E^{(k)}(t)$, where λ_k and $E^{(k)}(t)$ correspond to the Lagrangian multiplier and resource consumption at node i for $1 \leq k \leq M$, and M is the number of nodes involved in the scheduling.
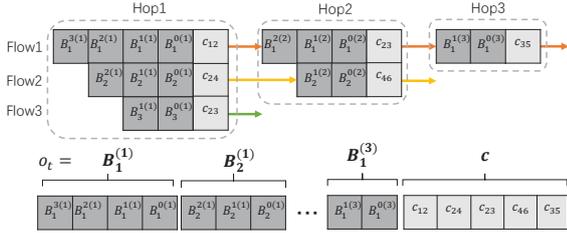

Figure 5: Node-level Merging of a three-flow case. Flow i' node j is associated with a buffer B_j^i. Flows are aligned with starting nodes then concatenated as observation o_t.

A common approach for multi-hop scheduling is to address the scheduling problem in each node separately. However, in delay-constrained multi-hop networks, scheduling each node separately cannot satisfy the hard delay constraint. Multi-agent architecture [33, 35] has been adopted for multi-hop networks but with more computation resource requirements and is hard to train. Node-level merging provides a novel way of concatenating buffers of different hops in flows together, making its training similar to that for a single-hop scheduling problem except with a higher input dimension. This avoids training multiple agents for multi-hop networks, and efficiently reduces the complexity in training. It allows RSD4 to perform well and outperforms other methods, even when the system scale in terms of state dimension is significantly increased in multihop networks (see Section 6.5).

5 PROPOSED METHOD: RSD4

We present our novel Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient (RSD4) algorithm in Algorithm 1. RSD4 builds upon the recurrent deterministic policy gradient [8] and softmax deterministic policy gradient [22], and introduces several novel components for handling the scheduling problem in partially observed settings. Specifically, RSD4 is a model-free DRL algorithm, which takes the popular actor-critic [12] framework. It well handles partial observability issues by memory mechanism enabled by recurrent neural networks (RNNs) and resolves the overestimation problem in existing recurrent DRL methods with the softmax operator. Thus, it owns advantages from recurrent deterministic policy gradient [8] and softmax deterministic policy gradient [22]. RSD4 also adopts a double-branch architecture from [24] to better utilize the memory mechanism of the RNN, and implements a delayed policy update frequency to further reduce the variance in value estimate.

Initially, RSD4 makes use of the state-action function $Q(h_t, a_t; \theta)$ parameterized by θ, which is defined as

---

Algorithm 1 RSD4 with decomposition

Require: λ_0, resource limit E_0, learning rate α, Precision δ, episode number M, episode length T, batch size b, target update rate τ.

1: $\lambda_1 \leftarrow \lambda_0, \lambda_0 \leftarrow 0, k \leftarrow 1$
2: **while** $|\lambda_k - \lambda_{k-1}| > \delta$ **do**
3:     Initialize N learning environments with $r_t^{(i)} = \beta_i d_i(t) - \lambda e_i^\top(t) B_i(t)$ for i-th environment.
4:     Initialize critic networks Q_1, Q_2, and actor networks π_1, π_2 with random parameters $\theta_1, \theta_2, \phi_1, \phi_2$. Initialize target network $\theta_1^- \leftarrow \theta_1, \theta_2^- \leftarrow \theta_2, \phi_1^- \leftarrow \phi_1, \phi_2^- \leftarrow \phi_2$.
5:     Initialize replay buffer $\mathcal{D}$.
6:     **for** episodes = 1 to M// Episodic interaction **do**
7:         **for** $t = 1$ to T **do**
8:             **for** $i = 1$ to N **do**
9:                 Receive sub observation o_t^i
10:                 $h_t^i \leftarrow h_{t-1}^i, a_{t-1}^i, o_t^i$
11:                 Select action a_t^i based on π_1 and π_2.
12:             **end for**
13:         **end for**
14:     Store $(o_1^i, a_1^i, r_1^i, ..., o_T^i, a_T^i, r_T^i)$ in $\mathcal{D}$ for $i = 1$ to N.
15:     **for** $i = 1, 2$ // Double learning **do**
16:         Randomly sample a batch of b episodes: $\mathcal{B} = \{(o_1, a_1, r_1, ..., o_T, a_T, r_T)\}$ from $\mathcal{D}$.
17:         **for** $t = 1$ to T // Recurrent softmax learning **do**
18:             Sample K noise $\epsilon \sim \mathcal{N}(0, \sigma')$
19:             $\hat{a}_t \leftarrow \pi_i(h_t; \phi_i^-) + clip(\epsilon, -c, c)$
20:             $\hat{Q}(h_t, \hat{a}_t) \leftarrow \min_{j=1,2}(Q_j(h_t, \hat{a}_t; \theta_j^-))$
21:             Compute softmax$_\beta(\hat{Q}(h_t, \cdot))$ by Eq. (8)
22:             $y_t \leftarrow r + \gamma(1 - d)$ softmax$_\beta(\hat{Q}(h_t, \cdot))$
23:         **end for**
24:         Update θ_i according to Bellman loss: $\frac{1}{N} \sum_{h \in \mathcal{B}} \sum_t (Q_i(h_t, a; \theta_i) - y_i)^2$
25:         **if** episodes mod $d = 0$ // Delayed update **then**
26:             Update actor ϕ_i by recurrent policy gradient:

$$\nabla_{\phi_i} J(\phi_i) = \frac{1}{N} \sum_{h \in \mathcal{B}} \sum_t \left[\nabla_{\phi_i} (\pi(h_t; \phi_i)) \right.$$
$$\left. \nabla_a Q_i(h_t, a; \theta_i)|_{a=\pi(h_t; \phi_i)} \right]$$

27:             Update target networks: $\theta_i^- \leftarrow \tau\theta_i + (1-\tau)\theta_i^-, \phi_i^- \leftarrow \tau\phi_i + (1-\tau)\phi_i^-$
28:         **end if**
29:         **end for**
30:     **end for**
31:     Obtain policy π_k and evaluate E_{π_k}.
32:     $\lambda_k = \lambda_k + \alpha(E_{\pi_k} - E_0)$ // Gradient update
33:     $\lambda_{k-1} \leftarrow \lambda_k$
34: **end while**
35: Output π_k

$$Q(h_t, a_t; \theta) = \mathbb{E}_{s_t, a_t, ..., s_{t+T}, a_{t+T}|h_t, \pi(\cdot; \phi)} \left[\sum_{i=0}^{T} \gamma^i r(s_{t+i}, a_{t+i}) \right],$$
$$(6)$$

where the expectation is taken with respect to the conditional probability $p(s_t, a_t, ..., s_{t+T}, a_{t+T}|h_t, \pi(\cdot; \phi))$ of the trajectory distribution associated with history h_t and the policy $\pi(\cdot; \phi)$. RSD4 initializes

double critic networks and double actor networks, where critic networks $Q(h_t, a_t; \theta)$ estimate the value of state-action pairs, and actor networks $\pi(\cdot; \phi)$ are responsible for outputting control actions.

5.1 The Training Algorithm

Recurrent Network Architecture. Despite the success of RL in a number of challenging tasks, state-of-the-art RL algorithms such as TD3 [6] are limited to solving fully-observable tasks. As a result, they can fail when faced with partially observable tasks as the problem considered here. To address this problem, compared to prior non-recurrent policy gradient methods, RSD4 incorporates recurrency in designing the architecture for neural networks rather than simply using feedforward networks in the policy update. This strengthens the memory capability of RSD4, and enables it to learn hidden factors or temporal correlation of the system dynamics. We then update the policy by RDPG [8]:

$$\nabla_\phi J(\pi(\cdot; \phi)) = \mathbb{E}_{h_t}\left[\sum_t \nabla_\phi(\pi(h_t; \phi)) \nabla_a Q(h_t, a; \theta)\Big|a = \pi(h_t; \phi)\right]. \tag{7}$$

Here to compute the RDPG, a sequence of episodes will be stored in the replay buffer $\mathcal{D}$ as a training sample (line 14). RSD4 computes target values $(y_1, y_2, ..., y_T)$ for each sampled episode using the recurrent networks in lines $17 - 23$. Critics and actors are updated recurrently, as shown in line 24 and 26, respectively.

Softmax Double Learning. Having a good estimate of the value function is critical for RL agents to achieve good performance [22]. We therefore propose to incorporate the softmax operator for more accurate value function estimation in different scenarios. In lines $21 - 22$, we compute the Q-function by softmax operator by:

$$\text{softmax}_\beta(Q(h, \cdot)) = \frac{\mathbb{E}_{a\sim p}\left[\frac{\exp(\beta Q(h,a;\theta))Q(h,a;\theta)}{p(a)}\right]}{\mathbb{E}_{a\sim p}\left[\frac{\exp(\beta Q(h,a;\theta))}{p(a)}\right]} \tag{8}$$

where β denotes the inverse temperature and p is the sampling distribution. The target value for critic Q_i is given by $y_i = r + \gamma \, \text{softmax}_\beta(\hat{Q}(h, \cdot))$ in line 22, where $\hat{Q}(h, \cdot)$ denotes the value estimation function in line 20.

RSD4 further adopts a delayed policy update mechanism from [6] (line 25) to avoid training divergence due to frequent updates of the policy. Thus, the policy network is updated at a lower frequency than the value network to minimize error before introducing a policy update, with the similar goal of making the scheduling policy more robust in various system dynamics.

5.2 Network Architecture

RSD4 uses double actor-networks and double critic-networks. The architecture of critic networks in Figure 6, and actor networks are similar, with the difference being removing the action a_t in the input and changing the output value $Q(s_t, a_t)$ to action a_t. The recurrent layers build upon Long-Short-Term-Memory (LSTM) [9] to perform RDPG in Eq. (7), and there are two parallel branches, i.e., the fully connected branch and the LSTM branch. This architecture is firstly proposed in [24] and is effective for our scheduling problem, as validated in our experiments.

The LSTM branch is designed to strengthen the memory ability of RSD4 algorithm, since it allows the agent to incorporate a large amount of network measurement history into its state space

to capture the long-term temporal dependencies of actual system dynamics. The LSTM layer is embedded in the second layer of the multilayer perceptron feature extractor. Consequently, our RSD4 algorithm can well handle various partially observable settings. The fully connected branch is designed to capture more information and improve expressiveness in the current time slot, which provides subsequent layers with more direct access to the current state without requiring information to filter through the LSTM branch. This makes RSD4 more sensitive to the current system state, so that abrupt changes of environmental conditions, e.g., a sudden burst of arrivals or temporal channel condition degradation, can be detected rapidly (shown in Section 6.4.3).
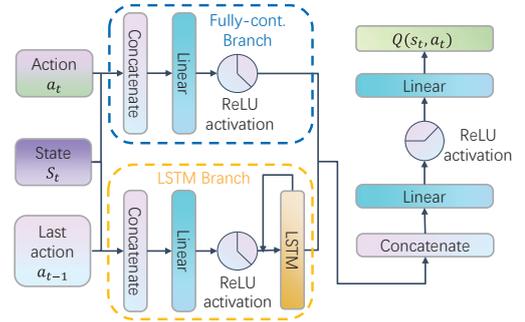

Figure 6: The architecture of the critic network. There are double parallel branches of the LSTM branch and the fully-connected branch, which are later concatenated together by a fully-connected layer to output Q value.

REMARK 3. *The architecture in Figure 6 accounts for the ability to resolve partial observability issues. Most non-DRL-based methods and non-recurrent DRL algorithms do not directly handle this case. Our POMDP formulation and RSD4 solution well handle such potentially non-stationary and partially observable dynamics by the memory mechanism enabled by LSTM, which helps the agent to learn from a batch of history and reduce the impact of temporal variability.*

6 EXPERIMENT RESULTS

We conduct extensive experiments on RSD4 based on real-world datasets and simulated data. We first present an ablation study of RSD4 and then compare performances of RSD4 with existing DRL methods and classical non-DRL-based algorithms. Then, we design various hard settings to validate the ability of RSD4 to handle partially observability and scalability. Each experiment is repeated with 5 different random seeds, and the average result is presented.

6.1 Environment Setup

Below, we specify the single-hop environment based on the network in Figure 2. The multi-hop environment is constructed in a similar way and will be specified in Section 6.5.

Arrivals. The arrivals of different users are given by a LTE dataset [15], which records the traffic flow of mobile carriers' 4G LTE network in approximately one year. We construct an environment with four types of arrivals given by the LTE dataset, which are visualized in Figure 7(a). The character of selected data records is given in Table 1, which simulates four representative tasks, i.e., file transmission, online forum, VR gaming, and text communication, according to their rates and delay requirements.
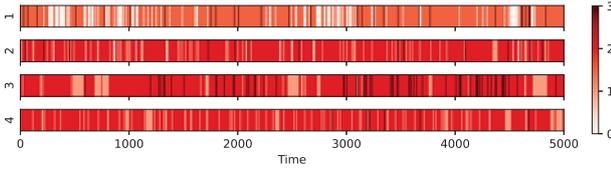
Service Channel Conditions. The channel states are given by a wireless 2.4GHz dataset [30], which samples the received signal strength indicator (RSSI) in the check-in hall of Shenzhen Baoan International Airport. Each channel state is quantized into 4 states, each representing a RSSI level, which is visualized in Figure 7(b), whose average channel states are given in Table 1.

Table 1: Summary of arrivals and channel states.

User	Rate	τ_i	Character	Avg. Channel
1	1.96	6	File Transmission	1.79
2	0.91	6	Online Forum	1.83
3	2.46	1	VR Gaming	1.82
4	0.70	1	Text Communication	1.77
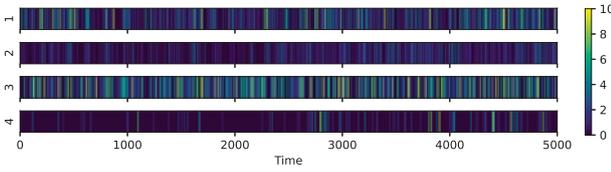
(a) Visulization of four arrivals in 5000 time slots.

(b) Visulization of channel conditions in 5000 time slots.

Figure 7: Heat maps of arrivals and channel conditions.

Service Outcomes. The probability that a transmission for user i is successful under channel state c and allocated resource e is

$$P_i(e, c) = \frac{2}{1 + \exp[-2e/(f_i^3 c)]} - 1 \qquad (9)$$

as that in [3], where f_i denotes the distance between user i and the server. This models a wireless downlink system.
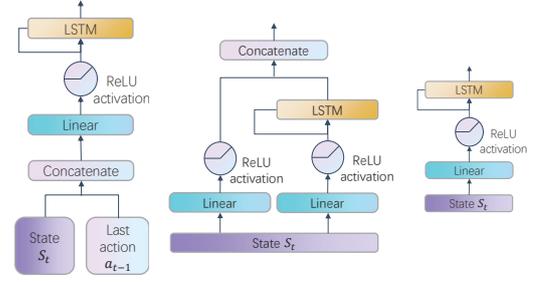
6.2 Ablation Study

We first present an ablation study on network architectures and important hyperparameters including policy update frequency and learning rate in this subsection. The results will guide us in our algorithm parameter setting. To keep the underlying MDP well-defined with time-invariant distributions, in the ablation study, we use simulated arrivals and channels in the environment by pre-specifying their distributions.

6.2.1 Network Architectures. We compare different network architectures in Figure 8 with our architecture in Figure 6.[3] These architectures are tested with RSD4 on the task of maximizing the Lagrangian function in Eq. (2) on $\lambda = \{0, 0.1, ..., 1\}$, since we find this range of λ is the region of possible optimal multiplier λ^* for our environment setting. We normalize the rewards obtained under different λ and results are shown in Figure 9(a), where rewards obtained with user-level decomposition mentioned in Section 4.2 are

---

[3]We only present the first several layers of the critic network, while the last two layers are the same as that in Figure 6.

presented as well. We find that double-branch architecture obtained maximum rewards in both situations, and rewards are further improved by taking the last action a_{t-1} as input to LSTM (see Figure 8). This validates the advantage of our proposed double-branch architecture and implies that taking action a_{t-1} in the last timestep into account can better capture hidden correlation across time.

(a) Single-branch (b) Double-branch without a_{t-1} (c) Single-branch with a_{t-1} without a_{t-1}

Figure 8: Different Network Architectures (part).
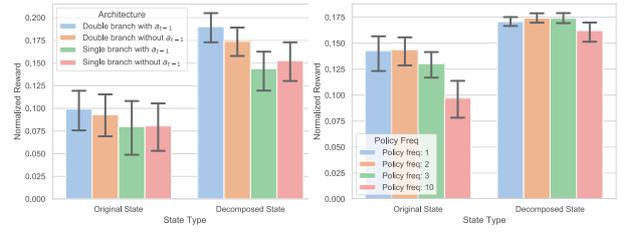
(a) Different network architecture　(b) Different policy frequencies

Figure 9: Experiments of the ablation study.

6.2.2 Policy Delay. RSD4 adopts a delayed policy update mechanism (line 25 in Algorithm 1), to avoid training divergence due to overestimating a poor policy [6]. The policy network is updated at a lower frequency than the value network to minimize error before introducing a policy update. We evaluate different policy frequencies on the same simulated environment as Section 6.2.1 in Figure 9(b). The results show that a moderate policy frequency of 2 improves peak reward in both original and decomposed cases.

6.2.3 Learning Rate. The learning rate α in Algorithm 1 is critical. We execute Algorithm 1 under different α values. Figure 10(a) and 10(a) show the iteration of λ_k and resource consumptions, respectively. A learning rate of 0.1 is too large such that $\{\lambda_k\}$ fluctuates largely, while a small learning rate e.g., 0.01 does eliminate the large fluctuation of $\{\lambda_k\}$ but convergences too slow. Thus, we use a decaying learning rate, which halves the learning rate when λ flips in three consecutive time slots, i.e., $\alpha_{k+1} = \alpha_k$ if $\lambda_{k+1} \geq \lambda_k \geq \lambda_{k-1}$ or $\lambda_{k+1} \leq \lambda_k \leq \lambda_{k-1}$, otherwise $\alpha_{k+1} = \alpha_k/2$.

6.3 Performance Comparison

We compare performances of RSD4 with existing DRL algorithms and classical non-DRL methods. The benchmark DRL algorithms include Twin Delayed DDPG (TD3) [6] and Softmax DDPG (SD3) [22], both are state-of-the-art algorithms. We apply them with Lagrangian dual to ensure average resource constraints. The non-DRL algorithms include Programming, Uniform, and Earliest Deadline First (EDF) [5]:

(a) λ_k (b) Resouce consumption
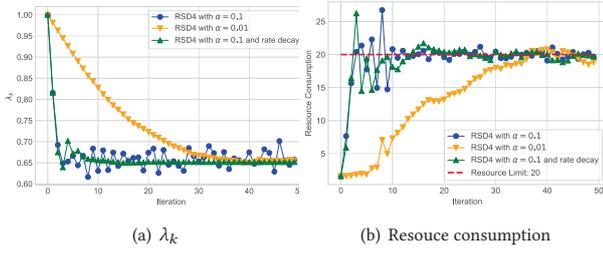
Figure 10: Iterations under different learning rates.

(i) Programming: Solve the following static constrained programming problem $\mathcal{P}_s$ for each time slot t with resource constraint using convex programming [27] (time index t omitted):

$$\mathcal{P}_s: \quad \max_{e} \quad \sum_{i=1}^{N} \beta_i \sum_{\tau=1}^{\tau_i} B_\tau^i P_i(e_\tau^i, c_{i,\tau}) \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} e_i^\top B_i \leq E_0$$

The optimal value $\mathcal{T}_{static}^*$ for $\mathcal{P}_s$ serves as the static optimal throughput for the Problem $\mathcal{P}$ in Eq. (1).

(ii) Uniform: Assign resources to different packets uniformly.

(iii) Earliest Deadline First (EDF): Assign all resources to packets in each user's queue with the shortest deadline equally.

To also evaluate the case when the resource expenditure at each time may be strictly bounded, we consider the hard resource constraint E_{max} for each time slot, i.e., scheduling decisions in each time slot cannot exceed E_{max}. We introduce two ways to ensure this. The first way is to scale each component equally of RSD4's decision to ensure the total expenditure is bounded by E_{max} (denoted as RSD4-E_{max}-1). The second way is to allocate resources to packets with smaller remaining time until the total expenditure is E_{max} and set all other allocations to zero (denoted as RSD4-E_{max}-2).

(a) Throughput   (b) Resource Consumption
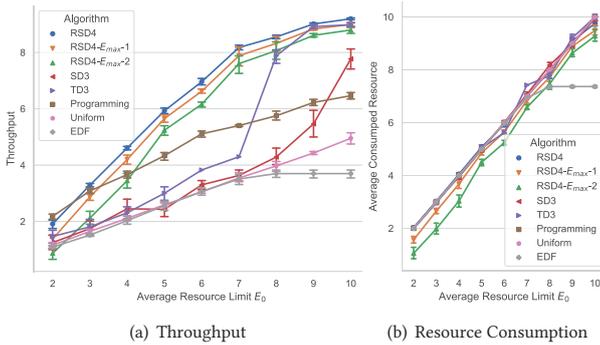
Figure 11: Comparison of different algorithms.

The comparison of RSD4 with other algorithms is shown in Figure 11 with $0 \leq E_0 \leq 10$, where arrivals and channel conditions are set according to real datasets summarized in Table 1. Here the channels are assumed to be unobservable to the scheduler. This scenario simulates a multi-user resource-constrained wireless base-station providing delay-constrained scheduling services. We find that performances of RSD4-E_{max}-1 and RSD4-E_{max}-2 are quite similar to RSD4 when setting $E_{max} = 2E_0$ in each group of experiments, which

means that hard resource constraint can be achieved with our methods without much performance degradation. Besides, from Figure 11(a), we see that RSD4 outperforms all benchmarks in each resource limit. In the small-medium resource regime with $0 \leq E_0 \leq 7$, the static optimality obtained by Programming ranks only lower than RSD4. However, in the large resource regime with $8 \leq E_0 \leq 10$, RSD4 and TD3 outperform classical methods significantly. From Figure 11(b), we see that all DRL algorithms satisfy the average resource limit, while EDF fails to fully utilize available resources since packets with the shortest deadline are too few to consume all available resources. Results in Figure 11 illustrate the superiority of RSD4 over the benchmarks.
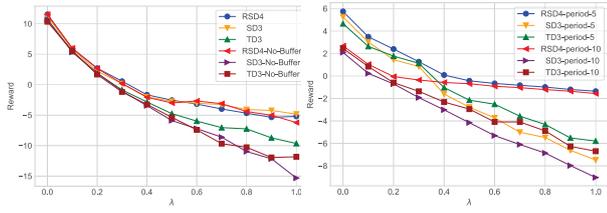
6.4 Partially Observable Systems

We compare the performance of RSD4 with other DRL algorithms to further validate the effectiveness of its recurrent module. Our comparison focuses on maximizing the Lagrangian function in Eq. (2) with the same environment of Section 6.3, which is essentially equivalent to the task of throughput maximization. In fact, each average resource limit E_0 corresponds to an optimal λ^* according to Section 3.2. Thus, maxmizing the Lagrangian function on some λ is equivalent to throughput maximization under some average resource limit of $E_0(\lambda)$ determined by λ. Thus, focusing on maximizing the Lagrangian function in Eq. (2) allows us to examine a wide range of problems parameterized by λ.

6.4.1 Missing Buffer State. We first consider an extreme environment where the buffer state is unobservable by the agent, and only arrivals and channels are given. Specifically, observation $o_t = [A(t), c(t)]$, which implies the agent needs to memorize arrivals and service outcomes across multiple time slots to have accurate estimations of current system states. From Figure 12(a), we find that with buffer state, i.e., the underlying MDP is fully observable, RSD4 and SD3 obtain similar maximum rewards under different λ. When the buffer state is missing, RSD4 still achieves almost the same maximum rewards, whereas non-recurrent DRL algorithms SD3 and TD3 suffer from significant performance loss.

6.4.2 Unobservable Hidden Factors. Another common type of partial observability comes from unobserved hidden factors in the underlying MDP, e.g., vehicle obstruction will influence in-tunnel wireless propagation channel which is hard to trace [29]. We design an environment where service outcomes are related to the time index, i.e., services are available only when the current time slot t is a multiple of some period, e.g., wireless communication interfered by periodic jamming signals. In this case, the underlying MDP is partially observable since the hidden factor, i.e., the period, is unknown to the agent and the observation is $o_t = [B^1(t), ..., B^N(t), c(t)]$. From Figure 12(b), we find that RSD4 outperforms TD3 and SD3 in both tested cases with periods 5 and 10, and the performance gain of RSD4 is more significant under large λ case, which corresponds to the small resource regime.

6.4.3 Time-varying Environments. We next investigate partial observability coming from the time variability of underlying system dynamics. We design two types of time variability by switching environment dynamics during the experiments, i.e., in one setting we double the rate of arrivals and in the other we change the channel statistics. In both cases the channel states are unobservable.

(a) Environment without buffer state.   (b) Environment with hidden factors.

Figure 12: Various partially observable settings.

The switch happens at slot $100, 000$ and results are shown in Figure 13 for the task of maximizing Lagrangian function under $\lambda = 0.5$, where the evaluated rewards of training processes under the environments with switching in the middle and switching at the very beginning. In both cases, we find that, soon after switching, RSD4 reaches the same rewards one can obtain by training in these environments from the very beginning, while TD3 and SD3 both suffer from reward loss after switching, even if they are trained from the beginning for this new environment. This validates the robustness and performance of RSD4 in time-varying environments.
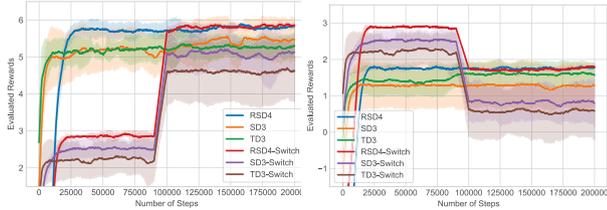
(a) Switch on arrival strength   (b) Switch on channel availability

Figure 13: Experiments on switching environments.

REMARK 4. *Experimental results on these three partially observable environments show the superiority of RSD4 at addressing partial observability issues. They also show the benefits of adopting a POMDP formulation in practical scheduling problems, since partial observability can arise from many sources. This comparison also sheds light on understanding the sub-optimality of non-recurrent DRL algorithms or classical non-DRL methods in POMDP.*

6.5 Scalability by Decomposition and Merging

We now turn to investigate the scalability of the algorithm by conducting experiments in systems with a large number of users and with multi-hop structures. We focus on the setting where the channel states are assumed to be unobservable. We will show that the user-level decomposition and node-level techniques in RSD4 make it a highly scalable solution.

6.5.1 Large-Scale System. We present the evaluated rewards of RSD4 with other algorithms on the task of maximizing the Lagrangian function with $\lambda = 0.3$ on different numbers of users ranging from 4 users to 400 users in Figure 14(a). Here we generate arrivals and channels according to predefined random processes, so that system dynamics are known and we can get the optimal reward by DP. It is worth noting that computing the optimal induces extremely high computation overhead in large-scale tasks and requires accessing full system dynamics, and is not practical in real-world applications. We examine this case here emphasize the near-optimality and effectiveness of RSD4. All baselines are implemented with the same set of hyperparameters to ensure fairness.

From Figure 14(a), we find that when user number is less than or equals to 20, the performance of different baselines do not differ much, and all of them achieve near-optimal rewards. However, with an increasing number of users, all DRL algorithms, including RSD4 without decomposition fail, as well as Uniform and EDF. Besides, RSD4 always achieves the best and near-optimal performance regardless of the number of users. Figure 14(b) also shows the learning curve under 50 users (from which the performance of different baselines start to diverge significantly), where only RSD4 achieves the nearly optimal reward. We also see that RSD4 without state-decomposition fails in this case. This demonstrates the necessity of state-decomposition, without which the algorithm parameter amount will become too large and prohibits efficient training.

REMARK 5. *When the system scale is beyond the hypothesis dimension of the underlying neural network, e.g., when the user number is larger than 20 in Figure 14(a), the performance of DRL algorithms will degrade rapidly, which means neural networks with more hyperparameters will be required. However, with more hyperparameters, neural networks are much harder to train and require significantly more computational power. With state decomposition, in contrast, RSD4 efficiently controls the state dimension and still retains the near-optimal performance. This validates the effectiveness of our proposed user-level decomposition.*
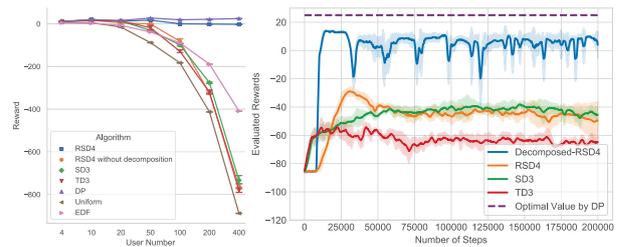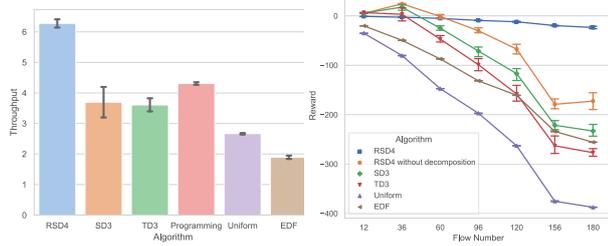

(a) Rewards on different user scales.   (b) Training process with 50 users.

Figure 14: Experiments on scalability.

6.5.2 Multi-hop Network. We now turn to the multihop network setting. Recall that we propose node-level merging in Section 4.3 to schedule under multihop networks. We conduct experiments on a multi-hop network depicted in Figure 4(a), where there are three paths, $Path_1 = 1 \rightarrow 2 \rightarrow 3 \rightarrow 5$, $Path_2 = 2 \rightarrow 4 \rightarrow 6$, and $Path_3 = 2 \rightarrow 3$. $Path_1$, $Path_2$, and $Path_3$ has 3, 4, and 5 flows passing through respectively. The 12 flows have different deadlines ranging from 3 to 5 timeslots and the scheduler needs to make decisions at nodes 1, 2, 3 and 4 simultaneously, with average resource limits 10, 30, 0.3, 3, respectively. The arrival and channel states are drawn from real-world datasets specified above, and the throughput obtained by different algorithms are shown in Figure 15(a).

We see that RSD4 achieves the maximum throughput and significantly outperforms other benchmarks, including TD3 and SD3 and non-DRL-based methods Programming, Uniform, and EDF. To further validate the scalability of our RSD4 algorithm, we fix the four multiplier values to 0.5 and compare it with other benchmarks on the task of maximizing the Lagrangian function for this multi-hop network. We test the algorithm with an increasing number of flows from 12 flows to 180 flows. RSD4 shows a larger performance gain over other algorithms as the number of flows increase and saves computation resources by unified training.

(a) Throughput of different algorithms on a multi-hop network with 12 flows.   (b) Rewards on different user scales of multi-hop networks.

Figure 15: Experiments on multihop networks.

7 CONCLUSION

This paper studies the problem of multi-user latency-constrained scheduling with average resource constraints. To tackle partial observability and scalability issues, we propose a novel DRL algorithm, RSD4, which is a data-driven method based on a POMDP formulation. RSD4 guarantees resource and delay constraints by Lagrangian dual and delay-sensitive queues, respectively. RSD4 successfully tackles partially observable issues with a recurrent network module. It also enables robust value estimation with the softmax operator, and introduces the user-level decomposition and node-level merging techniques to ensure scalability. We conduct extensive experiments on both simulated environments and real-world datasets. Our results show that RSD4 is robust to various system dynamics and partially observable settings, and significantly outperforms existing DRL/non-DRL-based benchmarks.

ACKNOWLEDGEMENTS

REFERENCES

[1] Dimitri Bertsekas, Angelia Nedic, and Asuman Ozdaglar. 2003. *Convex analysis and optimization*. Vol. 1. Athena Scientific.

[2] Rajarshi Bhattacharyya, Archana Bura, Desik Rengarajan, Mason Rumuly, Srinivas Shakkottai, Dileep Kalathil, Ricky KP Mok, and Amogh Dhamdhere. 2019. Qflow: A reinforcement learning approach to high qoe video streaming over wireless networks. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. 251–260.

[3] Kun Chen and Longbo Huang. 2018. Timely-throughput optimal scheduling with prediction. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2457–2470.

[4] Nhu-Ngoc Dao, Anh-Tien Tran, Ngo Hoang Tu, Tran Thien Thanh, Vo Nguyen Quoc Bao, and Sungrae Cho. 2022. A Contemporary Survey on Live Video Streaming from a Computation-Driven Perspective. *ACM Computing Surveys (CSUR)* (2022).

[5] Khaled MF Elsayed and Ahmed KF Khattab. 2006. Channel-aware earliest deadline due fair scheduling for wireless multimedia networks. *Wireless Personal Communications* 38, 2 (2006), 233–252.

[6] Scott Fujimoto, Herke Van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *ICML* (2018).

[7] Chaofan He, Yang Hu, Yan Chen, and Bing Zeng. 2019. Joint power allocation and channel assignment for NOMA with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications* 37, 10 (2019), 2200–2210.

[8] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. 2015. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455* (2015).

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[10] I-Hong Hou and PR Kumar. 2010. Utility-optimal scheduling in time-varying wireless networks with delay constraints. In *ACM MobiHoc*. 31–40.

[11] Longbo Huang, Shaoquan Zhang, Minghua Chen, and Xin Liu. 2015. When backpressure meets predictive scheduling. *IEEE/ACM Transactions on Networking*

[12] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[13] Cheng Li, Pihe Hu, Yao Yao, Bin Xia, and Zhiyong Chen. 2019. Optimal multi-user scheduling for the unbalanced full-duplex buffer-aided relay systems. *IEEE Transactions on Wireless Communications* 18, 6 (2019), 3208–3221.

[14] Chih-ping Li and Michael J Neely. 2013. Network utility maximization over partially observable Markovian channels. *Performance Evaluation* 70, 7-8 (2013), 528–548.

[15] Naebolo_Khanh Loi. 2018. Predict traffic of LTE network | Kaggle. https://www.kaggle.com/naebolo/predict-traffic-of-lte-network. Accessed: 2021-07.

[16] Lingling Lv, Chan Zheng, Lei Zhang, Chun Shan, Zhihong Tian, Xiaojiang Du, and Mohsen Guizani. 2021. Contract and Lyapunov Optimization-Based Load Scheduling and Energy Management for UAV Charging Stations. *IEEE Transactions on Green Communications and Networking* 5, 3 (2021), 1381–1394.

[17] Siqi Ma. 2017. Fast or free shipping options in online and Omni-channel retail? The mediating role of uncertainty on satisfaction and purchase intentions. *The International Journal of Logistics Management* (2017).

[18] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*. 197–210.

[19] Fan Meng, Peng Chen, Lenan Wu, and Julian Cheng. 2020. Power allocation in multi-user cellular networks: Deep reinforcement learning approaches. *IEEE Transactions on Wireless Communications* 19, 10 (2020), 6255–6267.

[20] Sangwoo Moon, Sumyeong Ahn, Kyunghwan Son, Jinwoo Park, and Yung Yi. 2021. Neuro-DCF: Design of Wireless MAC via Multi-Agent Reinforcement Learning Approach. *ACM MobiHoc* (2021).

[21] Yasar Sinan Nasir and Dongning Guo. 2019. Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks. *IEEE Journal on Selected Areas in Communications* 37, 10 (2019), 2239–2250.

[22] Ling Pan, Qingpeng Cai, and Longbo Huang. 2020. Softmax Deep Double Deterministic Policy Gradients. *Advances in Neural Information Processing Systems* 33 (2020).

[23] Su Pan and Yuqing Chen. 2018. Energy-optimal scheduling of mobile cloud computing based on a modified lyapunov optimization method. *IEEE Transactions on Green Communications and Networking* 3, 1 (2018), 227–235.

[24] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. 2018. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 1–8.

[25] Long Qu, Chadi Assi, Khaled Shaban, and Maurice J Khabbaz. 2017. A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks. *IEEE Transactions on Network and Service Management* 14, 3 (2017), 554–568.

[26] Ziv Scully, Mor Harchol-Balter, and Alan Scheller-Wolf. 2019. Simple near-optimal scheduling for the M/G/1. *ACM SIGMETRICS Performance Evaluation Review* 47, 2 (2019), 24–26.

[27] Kaiming Shen and Wei Yu. 2018. Fractional programming for communication systems—Part I: Power control and beamforming. *IEEE Transactions on Signal Processing* 66, 10 (2018), 2616–2630.

[28] Rahul Singh and PR Kumar. 2018. Throughput optimal decentralized scheduling of multihop networks with end-to-end deadline constraints: Unreliable links. *IEEE Trans. Automat. Control* 64, 1 (2018), 127–142.

[29] JiTing Song, Wei Wang, and Rashdan Ibrahim. 2021. The Impact of Obstruction by Vehicle on in-Tunnel Wireless Propagation Channel. In *2021 IEEE 4th International Conference on Electronic Information and Communication Technology (ICEICT)*. IEEE, 572–576.

[30] Wang Taotao, Xin Jiantao, Xu Wensen, Cai Yucheng, and Zhang Shengli. 2021. Wireless signal strength on 2.4GHz (WSS24) dataset. https://github.com/postman511/Wireless-Signal-Strength-on-2.4GHz-WSS24-dataset. [Online; accessed 17-2-2022].

[31] Liang Xiao, Yanda Li, Canhuang Dai, Huaiyu Dai, and H Vincent Poor. 2017. Reinforcement learning-based NOMA power allocation in the presence of smart jamming. *IEEE Transactions on Vehicular Technology* 67, 4 (2017), 3377–3389.

[32] Renchao Xie, Qinqin Tang, Chenghao Liang, Fei Richard Yu, and Tao Huang. 2020. Dynamic computation offloading in IoT fog systems with imperfect channel-state information: a POMDP approach. *IEEE Internet of Things Journal* 8, 1 (2020), 345–356.

[33] Chunmei Xu, Shengheng Liu, Cheng Zhang, Yongming Huang, and Luxi Yang. 2020. Joint user scheduling and beam selection in mmWave networks based on multi-agent reinforcement learning. In *2020 IEEE 11th Sensor Array and Multichannel Signal Processing Workshop (SAM)*. IEEE, 1–5.

[34] Han Zhang, Wenzhong Li, Shaohua Gao, Xiaoliang Wang, and Baoliu Ye. 2019. Reles: A neural adaptive multipath scheduler based on deep reinforcement learning. In *IEEE INFOCOM*. IEEE, 1648–1656.

[35] Fanqin Zhou, Lei Feng, Michel Kadoch, Peng Yu, Wenjing Li, and Zhili Wang. 2021. Multi-agent RL aided task offloading and resource management in Wi-Fi 6 and 5G coexisting industrial wireless environment. *IEEE Transactions on Industrial Informatics* (2021).

[24, 4 (2015), 2237–2250.]