Multi-Agent Reinforcement Learning Enabled Link Scheduling for Next Generation Internet of Things

Yifei Zou[a], Haofei Yin[a], Yanwei Zheng[1 a], Falko Dressler[b]

[a]*School of Computer Science and Technology, Shandong University, Qingdao, China*
[b]*School of Electrical Engineering and Computer Science, TU Berlin, Berlin, Germany*

Abstract

In next generation Internet of Things (NG-IoT) networks, numerous pieces of information are aggregated from the user devices and sensor nodes to the local computing units for further computing to support high-level applications. Those multitudinous transmission demands have raised new challenges for current link scheduling protocols. The centralized link scheduling protocols are inappropriate in some large-scale NG-IoT scenarios. The previously distributed link scheduling uses the randomized transmission scheme to avoid interference, making it hard to utilize the bandwidth resources fully. The multi-agent machine learning (MAML) technique is a potential approach to finding the most optimal link scheduling strategy. At the same time, the over-large state space will take a long time for them to approach the optimal solution, which reduces the practicality of the MAML. To fully utilize the bandwidth resource and improve the efficiency of link scheduling, this paper studies a multi-agent reinforcement learning enabled link scheduling problem. Different from the conventional MAML techniques that randomly select a state to do their exploration, in our multi-agent reinforcement learning algorithm, a *good* state is firstly obtained within polynomial time steps by executing a distributed and randomized sub-algorithm. We say a state is good if it is not far from the optimal state. Then, our multi-agent reinforcement learning scheme starts from the good state and does its exploration with an ε greedy scheme, which significantly reduces the time steps to get close to the optimal link scheduling strategy. Extensive simulations are conducted to investigate the performance of our work.

Keywords: multi-agent reinforcement learning, link scheduling, wireless network, Internet of Things network

1. Introduction

With the development of communication technology and Internet-of-Things (IoT) devices, the Next Generation Internet of Things (NG-IoT) has changed our life in its own way[1, 2]. Specifically, NG-IoT connects massive user devices and sensors to local units with some basic storage and computing resources (e.g., edge devices in edge computing) via the high-speed 5G/6G networks. So, a series of applications can be deployed on the units that are close to the data resource, which is a potential architecture to provide users with real-time and intelligent services. In the above framework, the fast transmission of the information packets is a fundamental issue, which can be formulated as a link scheduling problem from the view of the network layer. With an efficient link scheduling algorithm, the frequency and bandwidth resources in NG-IoT can be fully utilized to deliver the information packets so that the information can arrive at its destinations within a short time interval, which strongly supports those time-sensitive applications in NG-IoT.

Due to its importance and significance, a series of works have been conducted on link scheduling problems in the wireless environment. Generally speaking, the previous link scheduling works can be divided into two categories: centralized scheduling and distributed scheduling, according to their working mode. For centralized algorithms, such as [3, 4], the global information can be fully utilized to avoid conflicts and interference between links so that multiple links can be scheduled simultaneously at the same time. However, the centralized algorithm requires lots of global information, which is difficult to be deployed in large-scale networks. For distributed algorithms, such as [5, 6], the randomized transmission scheme can be used to obtain answers that are $O(\log^2 n)$-approximate and $O(\log n)$-approximate to the optimal solution, respectively. However, due to the stochastic nature of the algorithms, it is difficult for them to get closer to the optimal solution.

There are also some efficient works that solve the scheduling problem with the help of some machine learning [7, 8, 9] or heuristics techniques [10]. Compared with those randomized and distributed algorithms, the convergence property in ML helps the ML-based algorithm gets closer to the optimal solution for the link scheduling problem. Thus, the ML-based solutions can potentially have higher efficiency in delivering the information packets. However, all the above methods are performed under centralized conditions, where nodes must have prior global knowledge during training. No specific training node can have global information in the distributed premise. Solving the link scheduling problem by machine learning is essentially a multi-agent learning problem with a competitive relationship. In the multi-agent learning problem, each agent takes specific actions in its current state and is influenced by the states and actions of other agents in the environment to receive certain rewards. In the link scheduling problem, the mutual influence of multiple agents in this environment is particularly severe. For example, suppose a link is transmitting data when none of the other links also occupies the current wireless channel. In that case, the data will be delivered successfully. The communication will fail if it is surrounded by other links that are also using the wireless channel to transmit data. This phenomenon is also termed as the contention in the wireless channel. Suppose we use a single agent to decide whether each link should be sent at the current moment. In that case, the reward of that agent will be determined by a combination of its state, actions, and the state and actions of the surrounding nodes.

The global states of multi-agent learning methods grow exponentially with the number of nodes. In NG-IoT, consisting of numerous user devices and sensor nodes, the number of states will be too large to execute any centralized ML algorithm. Besides, the exploration of ML in the over-large state space will come across the following challenge. If a small exploration step is used on the over-large state space, the efficiency of exploration will be seriously insufficient, and the convergence speed will be slow or even cannot be converged. When the exploration step is too large, there will be a large amount of space that is not fully explored, and it is straightforward to miss the optimal solution or produce serious oscillations.

To overcome the above challenges that the state space in multi-agent machine learning is over-large while the exploration ability of current multi-agent ML techniques is limited, in this paper, we no longer let our multi-agent ML algorithm randomly select a state as the initial state, but use a distributed randomized algorithm to compute a *good* state as the starting point of our multi-agent ML. A good state means that the state of the nodes in multi-agent ML is not far away from the optimal solution, i.e., the exploration space has been limited. Then, we use the exploration capability of the reinforcement learning algorithm to explore the optimal solution around the good state. Fig. 1 is an illustration of the difference between the exploration space using a good state and a randomized state.

The detailed contributions of our paper are listed in the following:

- To address the problem that the state and action space becomes over-large for exploration in a multi-agent machine learning system with numerous learning agents, which results in a failure of convergence or a very slow convergence, our work proposes a new approach in which a good state is firstly computed by some approximation algorithm within a polynomial time steps as the initial state of the multi-agent ML. Then, a multi-agent reinforcement learning scheme is adopted to explore the optimal result based on the good state. Compared with the previous multi-agent ML that randomly selects an initial state, our work indicates that finding a good state as the starting point can reduce the exploration space and improve the performance of multi-agent ML on exploration. We hope that the new framework proposed in this paper can shed some light on designing multi-agent machine learning algorithms in some complex spaces.
- For the link scheduling problem under the next generation of IoT, we first formulate it into a multi-agent reinforcement learning problem. Then, a distributed and randomized algorithm is used to compute a good state for the following ML process. Finally, a multi-agent reinforcement learning algorithm is proposed, which starts from a good state and can get close to the optimal solution efficiently. Compared with the traditional stochastic algorithm, our link scheduling algorithm performs better because it can converge to the optimal solution stably.

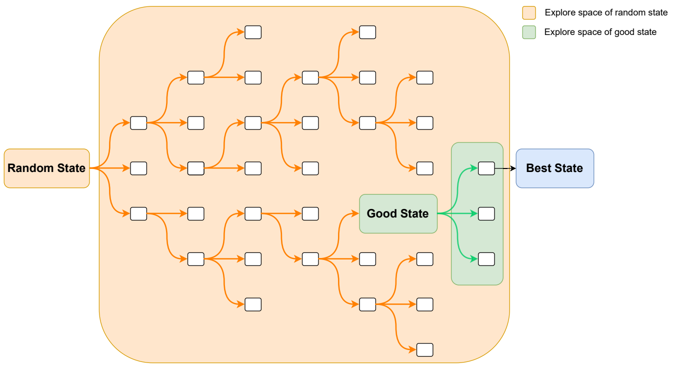


Figure 1: Good states have less space to explore than random states

The numerical simulations are presented to support our idea in this paper.

The remaining parts of this paper are organized as follows. In Section 2, the current work related to wireless link scheduling is introduced. In Section 3, the network model and the link scheduling problem are formulated. In Section 4, the multi-agent reinforcement learning algorithm is proposed in detail. In Section 5, extensive simulations are conducted to verify the algorithm's performance.

2. Related Work

The link schedule problem is usually transformed into a special problem of set partitioning. Formally, given a set L of links, the goal of the algorithm is to find a minimal division of L such that all links in each set after the division can complete their transmissions in the same time slice.

Initially, the study of link scheduling problems was often based on graph models, where the interference of one link to the rest of the links was often reflected in the edge or point weights in the graph. Related works are [11, 12, 13]. Subsequent studies had found that the graph model is difficult to model the wireless signal accurately, and the SINR model was first used in [14] to model the wireless link scheduling, which can deal more accurately with the effect on wireless interference on transmission. There are many subsequent works that also use the SINR model for link scheduling, and these works can be divided into two categories of centralized (e.g., [3, 4, 15, 16]) and distributed (e.g., [5, 6, 17, 18, 19]) algorithms in terms of working principle. In terms of computational complexity, [3] had proved that this problem is an NP-hard problem when the nodes use uniform power.

As for centralized algorithms. In [3], an approximation algorithm is proposed which maps each link to a unique $[2^k, 2^{k+1}]$ based on the length of the link, and then grids the deployed region based on the length of each of these links, and then tries to select as many links as possible in a grid to complete the scheduling. The time of this algorithm is $O(\log \Delta)$ approximated, where Δ is the ratio of the longest link to the shortest link. In [4], an approximation algorithm is proposed. The algorithm first sorts all the links by length, selects a smaller link each time, calculates the minimum distance between this link and the current set of already selected links, and then calculates whether the current link can be added to the set. The algorithm can achieve an approximate performance of $O(1)$. Since NG-IoT is often distributed and large-scale, numerous nodes are directly likely to be in relatively independent environments, which makes it difficult to apply these centralized algorithms on it.

As for distributed algorithms. In [5], the maximum average affectance coefficient is introduced to solve the competition problem in random, and nodes use a fixed probability to send data at each step according to the maximum average affectance coefficient, and this algorithm is $O(\log^2 n)$ asymptotically optimal. In [6], a distributed approximation algorithm is proposed, where the algorithm generates $\log^2 n$ equal probabilities and then lets all links enumerate any one of the probabilities several times, such that it can be shown that after the algorithm has finished running, there is a high probability that all links have finished scheduling at least once at the location of one of the probabilities. This algorithm is $O(\log n)$ asymptotically optimal. As mentioned above, most distributed algorithms are used to coordinate the interference between links by some adaptive stochastic process, but due to the natural uncertainty of stochastic algorithms, this makes the algorithms often have low performance and some distance between them and the optimal solution.

There are also some recent works that solve the link scheduling problem with machine learning (ML) techniques, such as [7, 8, 9]. Specifically, In [7], a geometric machine learning approach is used to solve the high mobility link scheduling in device-to-device(D2D) networks. The local graph around each D2D pair is modeled as a point by a set of regularized Laplace matrices on a Riemann manifold, and then the link scheduling decisions are classified in a supervised learning manner using the Riemann metric in a geometric support vector machine approach In [8], deep reinforcement learning is used to optimize resource allocation in software-defined networks by adaptive bandwidth allocation. In [9], a random forest classifier is used to control the millimeter wave radar to make optimal link selection by user's location and quality of service requirements. These algorithms solve not the classical link scheduling problem, but only some specific application problems containing the characteristics of the link scheduling problem. Moreover, these algorithms are not distributed algorithms, where all training is performed on the same device, and there is a lack of relevant research on distributed link scheduling for multi-agent learning.

Numerous studies have investigated the effectiveness of real-time learning through environmental information acquisition [20, 21, 22]. This approach has been found to enhance algorithm performance, including in the context of the multi-agent reinforcement learning we employ. Multi-agent reinforcement learning (MARL) has gained increasing attention in recent years as a promising approach to modeling and solving complex problems. In the context of reinforcement learning, the term "multi-agent" refers to scenarios where there are two or more independent agents that interact with the environment and potentially with each other. These agents' actions are influenced by the environment, and they may have varying degrees of cooperation or competition with each other [23]. As a result, MARL has been applied in various settings, ranging from games and robotics to communication and transportation systems. For example, some earlier studies have explored how MARL can help two agents with opposite goals to compete in a shared environment, while subsequent research has looked at the potential of this approach in addressing specific challenges in multi-terminal communication [24, 25, 26, 27]. It is worth noting that in multi-agent environments, the agents may have diverse options for their next actions. If we treat the space of all agents and their possible actions as a single entity, the size of this space will grow exponentially as the number of agents increases due to the combination of actions. However, a large action space can be disastrous for the algorithm's solution process, limiting the scalability of the number of agents to some extent [28].

3. Model And Problem Definition

We consider a NG-IoT scenario in which n nodes are arbitrarily deployed in a two-dimensional Euclidean space. V is the set of n nodes. For any two nodes i and j, we use $d_{i,j}$ to denote their Euclidean distance. Considering that the devices in NG-IoT always have their demands on information transmission, we use the set T to formulate such a demand. Specifically, if a node i has some information packets transmitted to j, we have $(i, j) \in T$, otherwise $(i,j) \notin T$. Define the size of the set of T to be m. Assume that from each node there is only one link with the current node as the sending node. For the case where the same node has multiple transmission requirements, a time-division multiplexing model can be used to complete its required transmissions at different time periods, which will not be discussed in this paper, in order to simplify our model and algorithm design.

3.1. Communication Model

The information packets are transmitted via a single hop wireless channel with physical interference constraint. All nodes transmit in a half-duplex mode, which means that a node can only listen to or send messages at one moment. The data transmitted between nodes is divided into packets of the same maximum size, so that all nodes can use the same time for single packet transmission, and we call the smallest unit of sending a single packet a time slot. According to the IEEE 802.11 protocol for wireless communication frames, a single wireless frame can hold a maximum of $p_m = 2346$ bytes of data. That is, for a data to be transmitted with a size of K bytes, exactly $\lceil \frac{K}{p_m} \rceil$ successful communications are required between nodes.

It is assumed that all communication occurs within a single channel. If there are multiple nodes sending messages simultaneously at the same moment, then there will be interference between multiple transmissions. We use the signal-to-interference-plus-noise ratio (SINR) model to measure whether the nodes' communication can be successfully proceed under interference when node u sends a wireless signal to v. Node v can receive messages from u if and only if :

$$SINR(u,v) = \frac{P/d_{u,v}^{\alpha}}{N + \sum_{i \in S/\{u\}} P/d_{i,v}^{\alpha}} \geq \beta \tag{1}$$

where N, α, β denote the global ambient noise, the signal fading factor, and an acceptable threshold determined by the physical device respectively, S is the set of nodes that sent messages in the same time slot. N is considered as a fixed constant in this paper. α is related to the environment in which the node is located and the physical medium in which the signal propagates, and can be considered as a constant in a fixed environment. β is also a constant determined by the hardware of the devices.

All nodes take a uniform power to transmit their information packet, which is determined by the transmission radius of the node. If the transmission radius is R, then the transmission power $P = cN\beta R^{\alpha}$, where $c \geq 1$ is a constant that is set with respect to the desired transmission rate of the network.

For reliable information transfer, any point-to-point data transfer requires that, in addition to the sender sending data to the receiver, the receiver should send an acknowledge (ACK) frame to the sender to confirm that the current message has been received. That is, for communication from node i to j, we consider a transmission successful and correctly completed if and only if j successfully receives the data frame sent by i and i successfully receives the ACK frame sent by j. Only if the sender successfully receives the ACK frame from the sender, the sender will continue to send the next data frames, otherwise the sender will assume that the current data frame was not successfully delivered and repeat the data frame again.

In addition to the basic ability to send and receive messages, a given node knows the constants needed in communication, including: N, R, α, β, c and $\hat{n}$, where $\hat{n}$ is a constant upper limit for the number of nodes and its value should be greater than the number of nodes n, and is available to all nodes.

3.2. Problem Definition

This paper focuses on how to maximize the throughput of continuous communication tasks and reduce the transmitting delay of nodes, so that more communication bandwidth is available between nodes or more nodes can be accommodated to complete the communication with limited radio frequency resources.

In general, we term the communication requirement from a node i to a node j as a link, denoted by $l_{i,j}$. Let's L be the set of link, and a link $l_{i,j} \in L$ if and only if $(i,j) \in T$. The problem is to choose $L_t \subseteq L$ for any time slots group t, where there are two time slots. For any $l_{i,j} \in L_t$, node i sends the data frame to j in the first time slot and j sends the ACK frame to i in the second time slot.

The optimization objective of the current problem is to maximize the sum of the available communication bandwidths of the nodes and to allow any link to maintain a relatively constant bandwidth over a continuous period of time. In other words, the current problem to deal with is a multi-objective optimization problem where our goal is to increase the overall bandwidth of the network as much as possible while keeping the link communication stable. And we expect to reach a Pareto Optimality solution.

Formally, we use the maximum delay to measure the stability of network communication. The maximum delay $lt_g(x)$ is defined as the maximum of the maximum delays of all links within a contiguous set of time slots of length x, which can be expressed as

$$lt_g(x) = \max_{(i,j) \in T} lt_{i,j}(x) \tag{2}$$

where $lt_{i,j}(x)$ denotes the maximum delay of the link $l_{i,j}$ within a contiguous set of time slots of length x. We define x time slot groups are numbered as $1, 2, \ldots, x$. In the x times of communication, we define the number of successful link transmissions $n_s(x)$ (abbreviated to n_s in the absence of ambiguity) and the time series $T_s(x) = \{t_1, \ldots, t_{n_s}\}$ at each successful transmission, where $\forall i \in [1, n_s), t_i < t_{i+1}$. Thus, the maximum delay of $l_{i,j}$ can be expressed as the following equation:

$$lt_{i,j}(x) = \begin{cases} \max\{t_1, x - t_{s_n}, \max_{i=1}^{i<n_s} t_{i+1} - t_i\} & n_s \geq 1 \\ x & n_s = 0 \end{cases} \tag{3}$$

In addition, we use the communication success rate of a link over time to measure the current bandwidth size of the network. The success rate of $l_{i,j}$:

$$sr_{i,j}(x) = \frac{n_s}{x} \tag{4}$$

In summary, the problem studied in this paper can be expressed by the following equation:

$$\begin{aligned} & \max_{(i,j)\in T} \ \tfrac{sr_{i,j}(x)}{lt_{i,j}(x)}, \\ s.t. & \begin{cases} L_t \subseteq L, & \forall t \in \mathbb{N} \\ x > 1, & \forall x \in \mathbb{N} \end{cases} \end{aligned} \tag{5}$$

3.3. Notations Table

The Table 1 explains the notations in this paper.

4. Multi-Agent Reinforcement Learning Approach

In this section, we present a multi-agent reinforcement learning algorithm that solves our defined link scheduling problem. Firstly, we clearly define the state in our multi-agent reinforcement learning algorithm. Then, we show that how to achieve a good state as the initial state of our multi-agent reinforcement learning algorithm. Finally, we show that how our multi-agent reinforcement learning algorithm proceeds.

4.1. Definition of state

As is illustrated in the SINR model, multiple nodes sending signals at the same time will interfere with each other, and too much interference will greatly reduce the success rate of communication. Constraining the number of sending nodes in the same time slot can effectively reduce the interference, so we use the probability p to control the sending of nodes, that is, for node i in $l_{i,j}$ in each time slot, i transmits with probability p and keep silent with the rest probability $1-p$.

Each node has its different environment for communication. some nodes have a large number of nodes nearby, while some have a small number, and the signal strength will be attenuated with the increase of distance, which means that the interference generated by the sender seems to be a “localized” phenomenon. When the number of nearby nodes is small, this means that even with a relatively high probability p, the receiving node suffers relatively little interference, which improves the communication efficiency of the link. When the number of nodes is large, the interference will prevent the nodes from receiving the signal properly, so it is important to set p smaller to reduce the global interference and ensure the success rate of communication. Therefore, the personalized p is chosen for each node in order to effectively improve global communication efficiency.

We define that the current communication probability p_x used by node x as the current state st_x of the node in the algorithm. Then, $S = \{st_1, \ldots, st_n\}$ is the global state of the whole multi-agent reinforcement learning system. Note that when our algorithm is executed, each node only knows its own state, and the global state is used only for better description and analysis. After clearly define the global state of our multi-agent reinforcement learning, we can see that the state space exponentially increases when the number of agents gets larger. If we randomly select a state to start the exploration, it takes a long time to find the optimal result. Thus, choosing an appropriate global state as the initial state for exploration is very important.

For a node, its state domain is a continuous space, i.e., $p \in (R)$. However, such a change has almost no effect on the network when the values of the two probabilities differ by a small amount. So we consider discretizing the probabilities to some extent, which reduces the number of states and thus speeds up the convergence. Specifically, we set a probability sequence P whose size is ds, where the ith element is defined as $p_i = 1/(4c_1^{i-1})$. In order that the probability of exploration can be sufficiently small, it should satisfy: $ds > \lceil \log_{c_1} \hat{n} \rceil$, where $1 < c_1 \leq 2$ is a constant that determines the sparsity of the chosen probabilities. For example, when $c_1 = 2$ and $\hat{n} = 8$, the generated probability sequence should be $1/4, 1/8, 1/16, 1/32$.

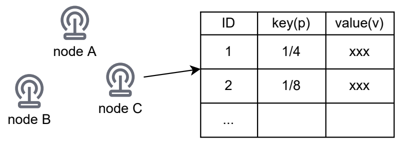


Figure 2: Example of weight dictionary

Each node maintains a dictionary of weights, where the key of the dictionary is each probability value in the above probability sequence and the value is the estimated value corresponding to this weight. The current dictionary has ds elements, which are listed in order of key from largest to smallest as $\{p_1, p_2, \ldots, p_{ds}\}$, and the corresponding value as $\{v_1, v_2, \ldots, v_{ds}\}$.

4.2. How to achieve a good state

The global state is a permutation of the states of each node, so the number of global states grows exponentially with the number of nodes, which means that the algorithm will have to explore an immense space, which greatly reduces the probability to find a good state, thus making the exploration process take a long time to reach a state that transmits information efficiently.

In order to find an approximately optimal link scheduling scheme quickly and with high probability to guarantee that each link can be scheduled once in a certain time, we use an $O(\log n)$-approximate algorithm to find a good state as the initial state of our multi-agent reinforcement learning.

Suppose the current node is u and the target receiver node is v. The ξ is an independent and uniformly distributed random variable with values $[0, 1]$, and the function $R(\xi)$ can sample the current random variable once.

Notation items	Description
n	Number of communication nodes in the network
V	The set of communication nodes in the network
$d_{i,j}$	The Euclidean distance between node i and node j
T	The demands on information transmission of the nodes
N	The global ambient noise
α	The signal fading factor
β	Acceptable threshold determined
R	the transmission radius of the nodes
P	The Euclidean distance between node i and node j
c	Constants related to network transmission rate
$l_{i,j}$	Communication requirement from node i to node j
L	The set of link
$lt_g(x)$	The maximum of the maximum delays of all links within a contiguous set of time slots of length x
$lt_{i,j}(x)$	The maximum delay of the link $l_{i,j}$ within a contiguous set of time slots of length x
$n_s(x)$ or n_s	The number of successful link transmissions within a contiguous set of time slots of length x
$sr_{i,j}(x)$	The success rate of $l_{i,j}$ within a contiguous set of time slots of length x

Table 1: Notations

The pseudo-code of the distributed algorithm is shown as Algorithm 1, where c_0 is a parameter used to guarantee a high probability of success of the algorithm.

Algorithm 1 Initialization algorithm

```
1:  $s_n = 0, T = \{\}$
2:  for $p \in P$ do
3:      for $i \in \frac{4}{p}c_0 \ln n$ time slots groups do
4:          if $R(\xi) < p$ then
5:              Send data in the first time slot
6:              Listen to ACK in the second time slot
7:              if Listened to the ACK from $u$ then
8:                  $s_n := s_n + 1$
9:                  $T := T \cup \{i\}$
10:             end if
11:         else
12:             Halt
13:         end if
14:     end for
15: end for
```

For node u, at the end of the algorithm run, the algorithm should populate the dictionary of weights based on the parameters obtained in the algorithm. In the initialization algorithm, the probability p_i will run $r_i = \frac{4}{p_i}c_0 \ln n$ times, thus, the value corresponding to this probability in the weighting dictionary should be:

$$v_i = \frac{sr_{u,v}(r_i)}{lt_{u,v}(r_i)} \tag{6}$$

The above algorithm is similar to the one in [6], with the difference that additional statistics on the parameters are needed in the current algorithm, and it has been shown that its algorithm has a high probability of successful completion. Algorithm 1 can be considered as an extension of the original algorithm and combined with the analysis of the original algorithm, in the problem of this paper, the parameters used in the original algorithm are a subset of the parameters used in the Algorithm 1, which means that the current algorithm additionally tries at some probabilities on top of the original algorithm to obtain more accurate probabilistic information. This does not affect the correctness and approximation of the algorithm.

In terms of application effects, the algorithm in [6] can only achieve $O(\log n)$-asymptotic optimality, and it is difficult to produce further optimization because of the randomization limitation, however the optimal state found in this way is only the starting state of the algorithm in this paper.

4.3. The MARL based on a good state

As is mentioned above, for each node, it already has a good but not the best state. Then, a ε-greedy reinforcement learning scheme can be used by each node to find a better solution in multiple iterations. The framework diagram of the algorithm is shown in Fig. 3.

Period of iteration. A continuous set of time slice groups d is used as the period of all nodes' single attempts, i.e., no node will change the currently selected p within a continuous set of d time slice groups. To guarantee the validity of the tries during this period, it should satisfy $d \geq 4N$. At the beginning of each iteration, a probability should be chosen in this iteration, and at the end, the values in the current dictionary will be updated using the data accumulated in this iteration.

Selection of probabilities. For each individual node, at the beginning of the iteration, the algorithm has the probability of ε to choose a probability currently considered optimal as the sending probability for the current time slice, and has the probability of $1 - \varepsilon$ to choose any key with

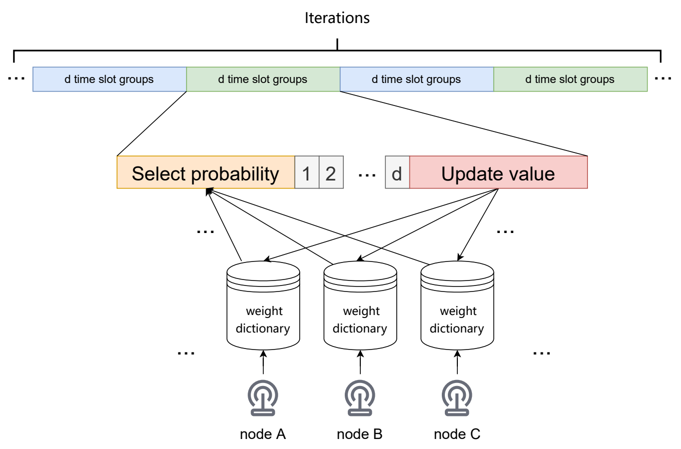


Figure 3: Diagram of our algorithm framework

Parameters	Description	Value
R	Communication radius	25
N	Global ambient noise	1
ε	Parameter in ε-greedy scheme	0.02
γ	Decay coefficient	0.95
$l_{\min}$	Minimum length of link	5
$l_{\max}$	Maximum length of link	20
c	Multiplier constant in communication power	2
d	Number of time slice groups in an iteration	800
c_0	Constant in Algorithm 1	2
c_1	Common ratio weight dictionary	1.25
m	Number of links	100
$\hat{n}$	Upper limit of nodes	200

Table 2: Default simulation parameter setting

Parameters	Description	Value
α	SINR signal fading parameter	$\{2, 4\}$
β	SINR receiving threshold parameter	$\{1.5, 2.5\}$
AL	Deployment area length	$\{100, 200, 400\}$

Table 3: Parameter settings used in the algorithm convergence experiments

equal probability in the current weight dictionary. Formally, the algorithm has the probability of ε, choosing

$$p_{\arg\max_{i=1}^{|P|} v_i} \tag{7}$$

and the probability of $1 - \varepsilon$ to choose

$$p_{\mathrm{DU}(1,ds)} \tag{8}$$

where $\mathrm{DU}(a, b)$ denotes a discrete uniform distribution in the interval $[a, b]$ with the existence of $a \leq b$ and $a, b \in \mathbb{N}$.

Update the values in the dictionary. In a round of iteration, a link $l_{i,j}$ needs to record the number of successful link transmissions n_s within the time series $T = \{t_1, \ldots, t_{n_s}\}$ at each successful transmission, where $\forall i \in [1, n_s), t_i < t_{i+1}$. If the probability of the current selection is p_k, the weight after updating would be:

$$v_k^{'} = \gamma \cdot v_k + \frac{sr_{i,j}(d)}{lt_{i,j}(d)} \tag{9}$$

where $sr_{i,j}(d), lt_{i,j}(d)$ is defined in Eq. 4, Eq. 3 respectively and $0 < \gamma < 1$ is the decay coefficient. A larger decay coefficient can better preserve the historical information, which is good for maintaining stability, and a smaller decay coefficient can better obtain the information of new iterations, which is good for fast convergence.

5. Simulation Experiments

In this section, a series of simulation experiments are conducted to evaluate the convergence of our algorithm, the superiority of the good state, the performance of the algorithm, and other metrics. The simulation program is written in C++ programming language, parallel accelerated using the OpenMP library, and runs on a computer with 32GB of memory, an Intel i9-12900k processor, and the Windows 11 operating system.

All the links are randomly generated in a square area of size AL$\times$AL. When generating, first a node is generated as the sender of the link, and then another node is randomly generated as the receiver in the optional area based on the minimum length $l_{\min}$ and the maximum length $l_{\max}$ of the link.

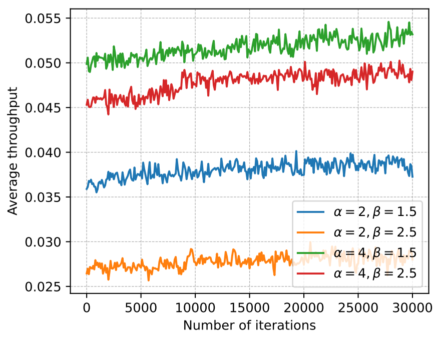
There are some general parameters set in the experiment. As shown in Table 2. In all subsequent cases, the program is executed with these parameters.

5.1. Convergence of the algorithm

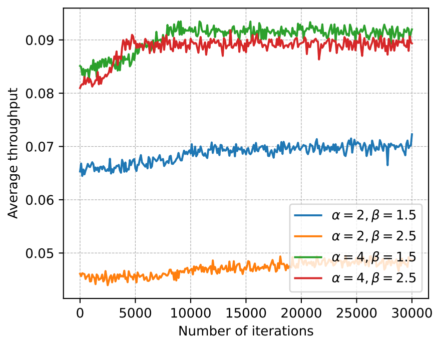
In this part, we study the convergence of our algorithm. Specifically, we mainly want to investigate whether our algorithm can converge to the optimal value in terms of average throughput and maximum latency when the SINR parameters α, β and node density vary. Here, we use Area Length (AL) to denote the side length of the square area where the node is deployed. Table 3 is the parameter setting in the current part of the experiment.

The results of the current part of the experiment are shown in Fig. 4 and Fig. 5. The horizontal axis of the graph represents the number of iterations of the algorithm.

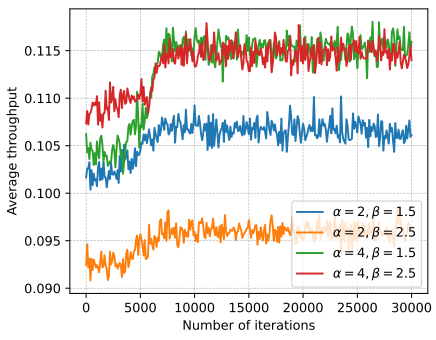
In Fig. 4, the vertical axis is the average throughput of all nodes in one round of communication. This figure has three subplots, each representing a different node density. The number of links is fixed here, what changes is the size of the area where the links are deployed. In each subplot



(a) Area = 100×100

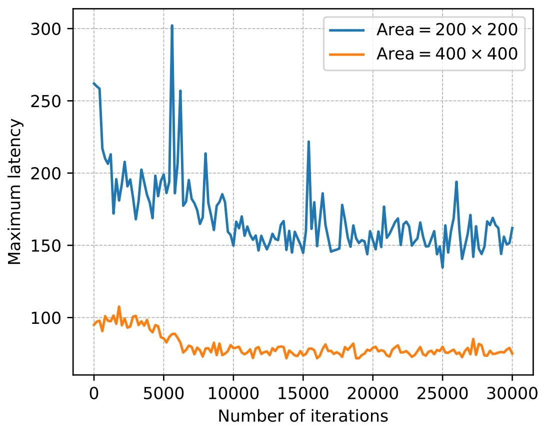


(b) Area = 200×200

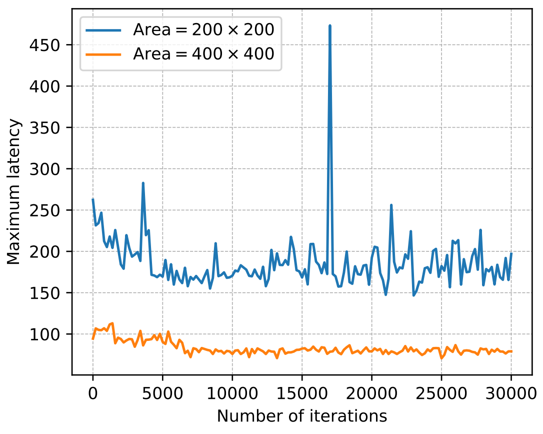


(c) Area = 400×400

Figure 4: Average throughput of the algorithm for different α, β



(a) $\beta = 1.5$



(b) $\beta = 2.5$

Figure 5: Maximum delay of the algorithm with the number of iterations for different β

there are four lines for different α, β, representing different network environments.

Observing the four curves of the three subplots in Fig. 4, we can see that when α is small, the interference generated in the environment becomes relatively strong, and the average throughput at this time is relatively small. Vice versa, it is rather large. This is consistent with the actual situation that the signal fading is enhanced when α is larger, and thus the interference is weakened. When β is larger, this receiving device is more sensitive to interference, so there is a lower average throughput than a smaller β, which is also consistent with the actual situation. From the subplots, we can see that our algorithm can improve the average throughput to some extent and converge when it reaches relative stability.

Compared with Fig. 4(a),Fig. 4(b), the enhancement of the reinforcement learning algorithm in Fig. 4(a) is relatively small. This is because, due to the large density of nodes, the distribution has a high probability of having more nodes within the communication radius of the node. The node is subject to interference located on a higher magnitude. At this time, the good state found by the initialization algorithm has been better selected for the probability of sending. Only a few local links are distributed more sparsely. The probabilities obtained by the initialization algorithm have space for further optimization, but for the global average throughput, this appears to be relatively small.

Comparing these three subplots, we can find that the algorithm can converge using a shorter number of iterations as the link distribution becomes sparse. Fig. 4(a), 4(b), 4(c) reach the optimal value and start converging at about 20000, 10000 and 8000 iterations, respectively. This is because, as the interference decreases, the link can better obtain the evaluation of the current transmission probability in the global environment rather than the evaluation failing due to severe interference. As the links become sparser, the average throughput of a single link becomes larger, which is realistic given the limited wireless resources of a single channel.

In Fig. 5, the vertical axis represents the maximum value of the maximum delay among all nodes within one iteration. This part of the experiment was performed when $\alpha = 4$ and $AL \in \{200, 400\}$, and two subplots with different β are represented.

Each subplot has two curves indicating the maximum latency in different network area. As seen from the two subplots, in the case of relatively sparse link distribution, the algorithm can effectively reduce the size of the maximum delay further based on the initialized algorithm maximum delay and keep it stable after a certain number of iterations. At AL = 200, where the links are relatively dense, a small amount of data with sudden growth in maximum latency occurs, such as in Fig. 5(a) at around 6000 iterations, and in Fig. 5(b) at around 17000 iterations. This is because a localized failure in the reinforcement learning exploration may heavily impact the behaviors of other nodes in the dense situation.

Parameters	Description	Value
α	SINR signal fading parameter	$\{2, 4\}$
β	SINR receiving threshold parameter	$\{1.5, 2.5\}$
AL	Deployment area length	$\{100, 200\}$

Table 4: Parameter settings used in good state experiments

Through these experiments, we have verified the convergence and effectiveness of our proposed algorithm when the network parameters vary.

5.2. *Effectiveness of good state*

As is mentioned in our algorithm design, starting the exploration from a good state makes our multi-agent reinforcement learning quicker to find the optimal solution. In this part, we will verify the above conclusion by conducting ablation experiments on the part of the initialization to find a good state. Suppose the existing initialization algorithm is not used at the beginning. In this case, the value is undefined in the weight dictionary of each node, and all undefined values are considered to be $-\infty$ when computing the maximum value. In other words, the exploration randomly starts from a state. TABLE 4 is the parameter setting in the current part of the experiment.

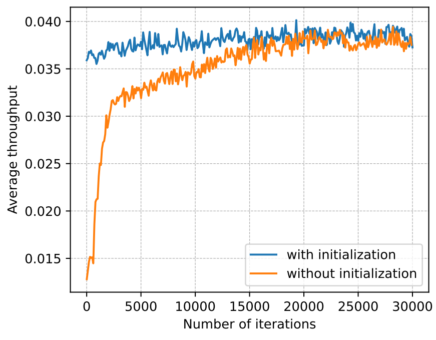
There are eight subplots in Fig. 6, each with a different deployment environment. There are two curves in each subplot, which indicate with and without initialization algorithm. The meanings of the horizontal and vertical coordinates in the figure are the same as those in the previous part and are not repeated here. Compared with the algorithm without initialization, the algorithm with initialization has the following points: (1) the algorithm using initialization can converge faster, shown in all subplots. (2) It can bring more throughput over a long time from the beginning of the algorithm, which is evident in 6(f), 6(h). (3) The algorithm with initialization can converge to higher throughput, which is evident in 6(b), 6(f).

Through the above experiments, we can find that in various environments, the algorithm with initialization converges to the best state more quickly than the algorithm without initialization, so our choice to use the initialization algorithm is very meaningful and can achieve great benefits.

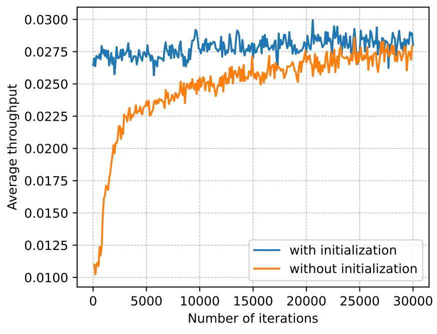
5.3. *Performance Analysis*

After finding the good state, the reinforcement learning algorithm can be used to make the chosen communication probability more justified and further reach a better state, thus achieving higher throughput and lowering the maximum latency.

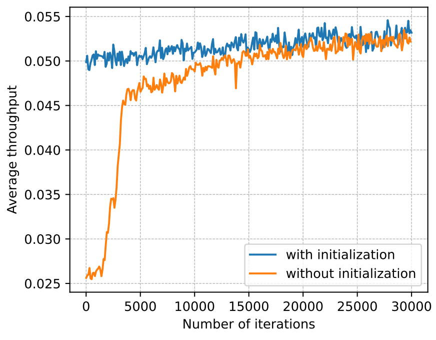
In this part, we study the performance of the reinforcement learning algorithm after finding a good state.



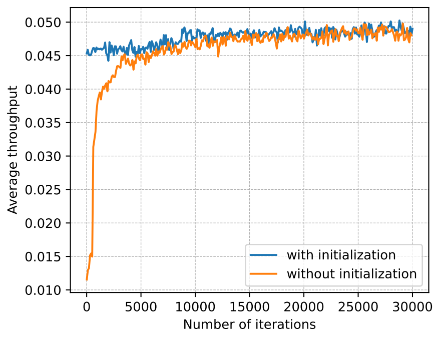
(a) $\alpha = 2, \beta = 1.5, \mathrm{AL} = 100$



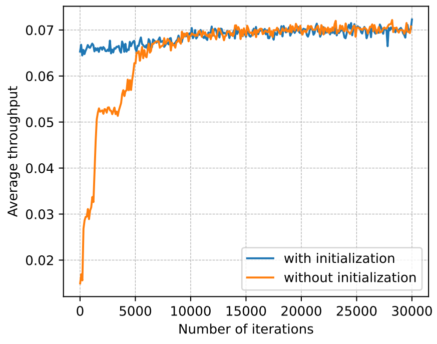
(b) $\alpha = 2, \beta = 2.5, \mathrm{AL} = 100$



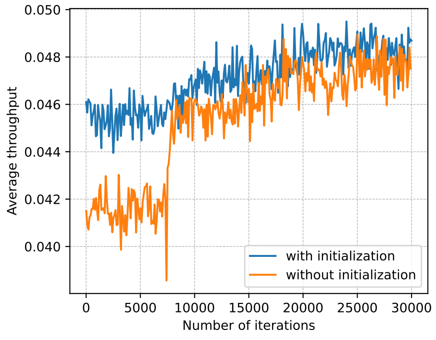
(c) $\alpha = 4, \beta = 1.5, \mathrm{AL} = 100$



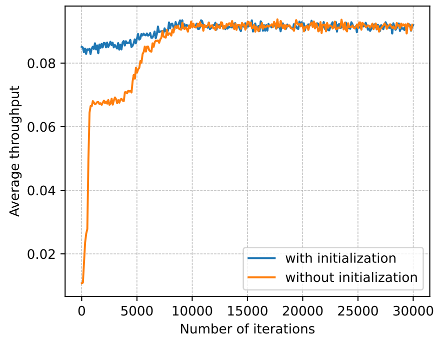
(d) $\alpha = 4, \beta = 2.5, \mathrm{AL} = 100$



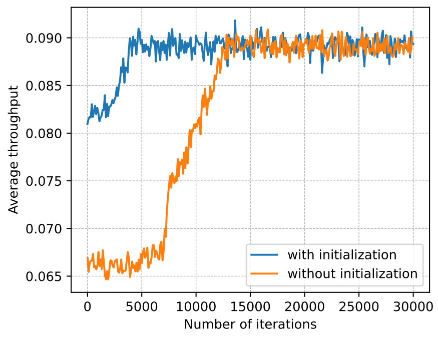
(e) $\alpha = 2, \beta = 1.5, \mathrm{AL} = 200$



(f) $\alpha = 2, \beta = 2.5, \mathrm{AL} = 200$

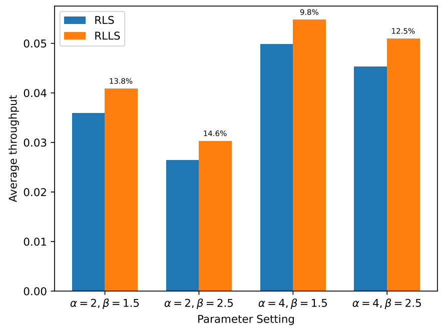


(g) $\alpha = 4, \beta = 1.5, \mathrm{AL} = 200$

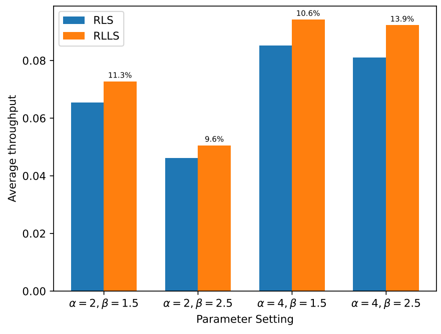


(h) $\alpha = 4, \beta = 2.5, \mathrm{AL} = 200$

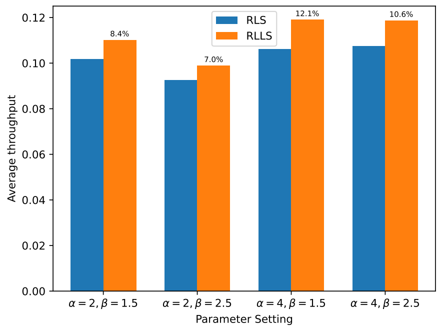
Figure 6: Effect of initialization under different $\alpha, \beta, \mathrm{AL}$ on the variation of the average throughput of the algorithm with the number of iterations



(a) $Area = 100 \times 100$



(b) $Area = 200 \times 200$



(c) $Area = 400 \times 400$

Figure 7: Performance statistics of average throughput for different $\alpha, \beta, \mathrm{AL}$

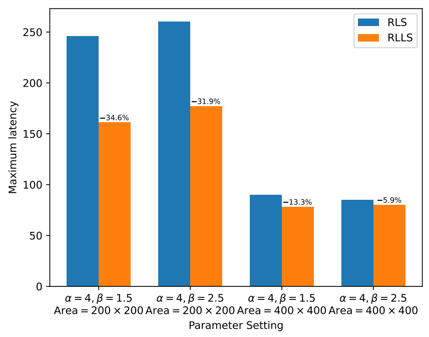


Figure 8: Performance statistics of maximum latency for different β, AL

We measure how much improvement can be achieved by continuously iterating and finding a better state compared to always using the good state.

The average throughput and maximum latency statistics are shown in Fig. 7, Fig. 8, respectively. The method always using the probability in good state are termed as the Random Link Scheduling (RLS) and the method use reinforcement learning to further explore the probability from the good state is termed as Reinforcement Learning Link Scheduling (RLRS). The two figures compare the performance of the above two methods.

It can be observed that reinforcement learning algorithms always deliver performance improvements of 7% to 14% in average throughput. The exact value that can be improved is related to the layout of the links in the randomly generated network. In some layouts, the good state found by the initialization algorithm is already close to the optimal solution, which leads to little room for further optimization afterwards. In terms of maximum latency, the algorithm can effectively reduce the latency per link by a maximum of between 6% and 35%. This can further ensure the fairness of transmission between links.

Through our experiments, we found that using reinforcement learning can indeed improve the throughput and reduce the latency of communication. The key to achieving these improvements is the continuous game among the nodes to find a transmission probability that can improve the average throughput and reduce the delay without causing significant interference to other nodes in the network. In this process, each node continuously adjusts its transmission probability based on the success rates and delays of its own transmission and other nodes' transmissions in the network. The algorithm encourages nodes to use a higher transmission probability as long as it does not cause significant interference with other nodes. Ultimately, the algorithm discovers the optimal transmission probability that maximizes channel utilization, and this is precisely why reinforcement learning algorithms are facilitating these improvements.

5.4. *Experimental conclusions*

Through extensive simulation experiments, we verified the convergence of the algorithm proposed in this paper and conducted ablation experiments on the initialization part of the algorithm to illustrate the effectiveness of the initialization part in the algorithm. After that, we further analyzed the algorithm's performance in the reinforcement learning part and counted the magnitude of the optimization in both the average throughput and the maximum delay after using the reinforcement learning algorithm. These simulation experiments provide rich data to support the effectiveness of our proposed algorithm above.

6. Conclusion

This paper studies a link scheduling problem in next generation Internet of Things networks with multi-agent reinforcement learning technique. Different from the traditional stochastic ones that approximate to the optimal solution with some factors, the machine learning technique is helpful in finding the most optimal solution. Whereas the multi-agent reinforcement learning technique cannot be directly adopted to solve the link scheduling problem since its state space exponentially increases when the number of agents gets larger, which results in an unacceptable exploring time. To overcome these challenges, in our multi-agent reinforcement learning algorithm, we first use a randomized sub-algorithm to obtain a good state within polynomial time steps. Then, our multi-agent reinforcement learning will do its exploration from the good state. Numerical simulations are conducted to show that the selection of a good state significantly helps in finding the most optimal state from the complex state space. Extending our method to some more complex network scenarios will be our work in the future.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62102232 and Natural Science Foundation of Shandong province under Grant ZR2021QF064.

References

[1] C. Chen, J. Jiang, Y. Zhou, N. Lv, X. Liang, S. Wan, An edge intelligence empowered flooding process prediction using internet of things in smart city, Journal of Parallel and Distributed Computing 165 (2022) 66–78.

[2] J. Gou, L. Sun, B. Yu, S. Wan, W. Ou, Z. Yi, Multi-level attention-based sample correlations for knowledge distillation, IEEE Transactions on Industrial Informatics (2022).

[3] O. Goussevskaia, Y. A. Oswald, R. Wattenhofer, Complexity in geometric sinr, in: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing, 2007, pp. 100–109.

[4] B. Huang, J. Yu, D. Yu, C. Ma, Sinr based maximum link scheduling with uniform power in wireless sensor networks, KSII Transactions on Internet and Information Systems (TIIS) 8 (11) (2014) 4050–4067.

[5] T. Kesselheim, B. Vöcking, Distributed contention resolution in wireless networks, in: International Symposium on Distributed Computing, Springer, 2010, pp. 163–178.

[6] M. M. Halldórsson, P. Mitra, Nearly optimal bounds for distributed wireless scheduling in the sinr model, Distributed Computing 29 (2) (2016) 77–88.

[7] R. Shelim, A. S. Ibrahim, Geometric machine learning over riemannian manifolds for wireless link scheduling, IEEE Access 10 (2022) 22854–22864.

[8] W.-x. Liu, J. Lu, J. Cai, Y. Zhu, S. Ling, Q. Chen, Drl-plink: Deep reinforcement learning with private link approach for mixflow scheduling in software-defined data-center networks, IEEE Transactions on Network and Service Management (2021).

[9] C. Tatino, N. Pappas, I. Malanchini, L. Ewe, D. Yuan, Learning-based link scheduling in millimeter-wave multi-connectivity scenarios, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.

[10] D. Yang, K. Gong, J. Ren, W. Zhang, W. Wu, H. Zhang, Tc-flow: Chain flow scheduling for advanced industrial applications in time-sensitive networks, IEEE Network 36 (2) (2022) 16–24.

[11] G. Sharma, R. R. Mazumdar, N. B. Shroff, On the complexity of scheduling in wireless networks, in: Proceedings of the 12th annual international conference on Mobile computing and networking, 2006, pp. 227–238.

[12] L. Jiang, D. Shah, J. Shin, J. Walrand, Distributed random access algorithm: scheduling and congestion control, IEEE Transactions on Information Theory 56 (12) (2010) 6182–6207.

[13] C. Joo, X. Lin, N. B. Shroff, Understanding the capacity region of the greedy maximal scheduling algorithm in multihop wireless networks, IEEE/ACM Transactions on Networking 17 (4) (2009) 1132–1145.

[14] T. Moscibroda, R. Wattenhofer, The complexity of connectivity in wireless networks, in: INFOCOM 2006: 25th Annual Joint Conference of the IEEE Computer and Communications Societies, Barcelona, Spain, 2006.

[15] J. Yu, B. Huang, X. Cheng, M. Atiquzzaman, Shortest link scheduling algorithms in wireless networks under the sinr model, IEEE Transactions on Vehicular Technology 66 (3) (2016) 2643–2657.

[16] T. Kesselheim, A constant-factor approximation for wireless capacity maximization with power control in the sinr model, in: Proceedings of the twenty-second annual ACM-SIAM symposium on discrete algorithms, SIAM, 2011, pp. 1549–1559.

[17] M. Andrews, M. Dinitz, Maximizing capacity in arbitrary wireless networks in the sinr model: Complexity and game theory, in: IEEE INFOCOM 2009, IEEE, 2009, pp. 1332–1340.

[18] D. Qian, D. Zheng, J. Zhang, N. B. Shroff, C. Joo, Distributed csma algorithms for link scheduling in multihop mimo networks under sinr model, IEEE/ACM Transactions on Networking 21 (3) (2012) 746–759.

[19] A. Fanghänel, S. Geulen, M. Hoefer, B. Vöcking, Online capacity maximization in wireless networks, in: Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures, 2010, pp. 92–99.

[20] Q. Hu, S. Wang, Z. Xiong, X. Cheng, Nothing wasted: Full contribution enforcement in federated edge learning, IEEE Transactions on Mobile Computing (2021).

[21] C. Wang, Q. Hu, D. Yu, X. Cheng, Online learning for failure-aware edge backup of service function chains with the minimum latency, arXiv preprint arXiv:2201.06884 (2022).

[22] C. Peng, Q. Hu, Z. Wang, R. W. Liu, Z. Xiong, Online learning based fast-convergent and energy-efficient device selection in federated edge learning, IEEE Internet of Things Journal (2022).

[23] K. Zhang, Z. Yang, T. Başar, Multi-agent reinforcement learning: A selective overview of theories and algorithms, Handbook of reinforcement learning and control (2021) 321–384.

[24] M. L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Machine learning proceedings 1994, Elsevier, 1994, pp. 157–163.

[25] J. Choi, S. Oh, R. Horowitz, Distributed learning and cooperative control for multi-agent systems, Automatica 45 (12) (2009) 2802–2814.

[26] J. Cortes, S. Martinez, T. Karatas, F. Bullo, Coverage control for mobile sensing networks, IEEE Transactions on robotics and Automation 20 (2) (2004) 243–255.

[27] D. Kim, S. Moon, D. Hostallero, W. J. Kang, T. Lee, K. Son, Y. Yi, Learning to schedule communication in multi-agent reinforcement learning, arXiv preprint arXiv:1902.01554 (2019).

[28] P. Hernandez-Leal, B. Kartal, M. E. Taylor, A survey and critique of multiagent deep reinforcement learning, Autonomous Agents and Multi-Agent Systems 33 (6) (2019) 750–797.