Deep Reinforcement Learning for Scheduling in Multi-Hop Wireless Networks

Invited Paper

Shuai Zhang, Bo Yin and Yu Cheng
Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616
szhang104@hawk.iit.edu, byin@hawk.iit.edu, cheng@iit.edu

***Abstract*—The efficient scheduling of transmission links in a wireless network with a certain optimization objective and subject to the interference and network flow constraints plays a central role in wireless networking research. As an alternative to traditional mathematical analysis, data-driven learning methods have shown promise in solving difficult problems by extracting knowledge from experiences and inspired applications of machine learning in wireless networking. In this paper, we focus on tackling the fundamental scheduling issue in multi-hop wireless networks with machine learning, facing the great challenges of the involvement of non-differentiable operations and the consideration of variable network topologies. To address these issues, we propose a reinforcement learning-based method to solve a class of network flow problems under the protocol interference model. Learning from experience, the proposed approach develops a strategy to sequentially select optimum subsets of links to transmit simultaneously to maximize the system throughput without causing interference. The model structure is designed in a way that incorporates network topological information to allow a flexible number of network nodes, and allows non-differentiable decision operation to pass informative gradient information. Experiments with synthetic and real-world deployment data demonstrate that the proposed algorithm achieves close-to-optimum performance at a significantly reduced time cost.**

I. Introduction

With the advent of high-capacity and low-latency next-generation communication networks, the need to efficiently use all available resources is more desired than ever. In existing wireless networks, a network optimization task is typically dealt with by a designated entity which solves a mathematical programming problem. Due to the interference among co-channel transmissions in a wireless context, which is reducible to the independence number problem, wireless network optimization is NP-hard in general [1], [2].

The recent success of machine learning in perceptual domains, e.g., computer vision or natural language processing, has marked a major paradigm shift in the development of algorithm design. In recent years, there have been attempts to harness the power of machine learning (ML) in networking problems such that smart decisions can be made by learning from a large number of past experiences [3]. While there are already some ML studies in physical layer [4], [5], network layer [6], [7], and the upper layers [8], [9], the application of ML in link scheduling (link layer) is quite limited. The existing scheduling studies based on machine learning are constrained to single-hop networking scenarios [10]. In this paper, we are to address, with a machine learning approach, the scheduling issue and network flow issue over a generic multi-hop wireless network in a joint manner.

In this work, we consider the canonical multi-commodity flow (MCF) problem in multi-hop wireless networks, where multiple flow demands between different source-destination node pairs exist. The central issue in multi-hop wireless networks is that different links in proximity, if using the same spectrum, interfere with each other when they transmit at the same time. Thus, the basic idea of scheduling is that the simultaneously active transmission links need to be far from each other; if network links are represented as nodes in a conflict graph, each set of active links form an *independent set* on the conflict graph. These different independent sets take turns to transmit in a time-sharing manner to satisfy the traffic demands. The MCF optimization over a multi-hop wireless network involves joint solution of routing and scheduling [1], [2].

The wireless MCF optimization with independent set based scheduling can be generically formulated as a linear programming (LP) problem for either single-radio single-channel multi-hop wireless networks [1] or more complex multi-radio multi-channel multi-hop wireless networks [2], [11]. However, the fundamental challenge is there are exponentially many possible independent sets, thus the wireless MCF optimization problem is still NP-hard in general. Existing approaches for addressing the scheduling mostly develop certain approximation algorithms that utilize graph theory or combinatorics [12], or heuristic algorithms based local search [13]. With the former, the drawback is that performance is sacrificed for theoretical performance guarantee in the worst or average case; the latter suffers from a high dependence on hand-crafted algorithmic parameters, which may be derived from an application setting different from the one it is used.

In this paper, we propose an innovative reinforcement learning (RL) based algorithm to solve the MCF optimization problem, with independent set based scheduling, in multi-hop wireless networks. Instead of applying the RL as a black-box end-to-end solver, our approach combines the power of neural networks with a proven algorithmic framework *delayed column generation* (DCG) [2], [14]. These studies show that the independent sets for scheduling can be searched and updated iteratively to keep improving the solution quality until the optimum is reached. This iterated structure of DCG inspires us that the solution procedure can indeed be formulated as a sequential Markov decision process (MDP), and thus enabling the development of RL based methods. Specifically, given the problem input, we treat it as an initial system state, to which the algorithm responds with its action. The action is exactly the selection of a set of communication links that can be activated simultaneously under the interference constraints, i.e., the selection of an independent set. Outcomes of the action will be evaluated with a reward, and the system state will also get updated according to the action and the current state. With an iterative operation which involves adjustment of action policies and some other interactions with the environment, the RL agent gives a collection of independent sets that are expected to be used in optimum scheduling.

We would like to emphasize that adopting RL to address the scheduling problem in MCF optimization is far more challenging than a straightforward application of standard RL implementation modules, with two fundamental reasons. First, the possible independent sets are exponentially many and unknown a priori, which thus leads to an exponentially large and discrete action space. However, existing RL methods perform best when there is a small, fixed number of discrete candidates to choose from or the space action is continuous, which is not applicable for our case. Second, it is desirable that the control policy learned by the RL agent can be applied robustly over diverse problem instances with a variable number of network elements and different network topologies. This prevents the straightforward use of top-of-the-shelf learning modules, because they typically assume the input and output variables have fixed dimensions.

As a response, we enhance the reinforcement learning framework with two significant new features that can facilitate our purpose. One is an order-invariant encoding module to describe the environment state, which can not only encode those independent sets already searched, but also represent network topology information in an order independent manner. Such an encoder equips our methods with the extendability, that is, the trained machine can be applied to different network topologies without retraining but can still maintain good performance. The other is the design of an action-searching mechanism to infer a proper action by incorporating non-differentiable operations into the decision process. In particular, a surrogate function that locally approximates the non-differentiable part while providing meaningful gradient information is used to guide the optimizer.

We evaluate our approach by using both synthetic examples and wireless networks from real world deployment. The performance data is collected by running a trained RL model to schedule the wireless link activation, and different network settings are used for testing the generalization ability. We find that the proposed method achieves an average of 26% of reduction in needed number of iteration to reach a performance level within 10% of the reference algorithm [2]. Moreover, such advantages are observed in problem instances that are differently generated from the ones used in training, suggesting a strong generalization ability to situations unseen before.

The main contributions of this paper are three-fold:

- First, we propose a reinforcement-learning approach that improves on the existing algorithmic framework. We combine the power of RL and a proven column generation framework to solve a complex network optimization problem, offering new insights for algorithm design and network improvement.
- Second, we design specific methods to tackle the varying network size and non-differentiable processing steps for solution generation, which are commonly encountered in networking scenarios but cannot be easily handled with the direct application of neural networks.
- Third, we test the performance numerically in terms of time-saving and solution quality across various network topologies and sizes to demonstrate the effectiveness of the proposed method. It is shown to achieve a fast convergence time and a high solution quality compared with other heuristic methods.

The remainder of the paper is structured as follows. The system model is presented in section II. In section III the formal description of the problem is given, followed by our algorithm design details. The numerical experiments' methods and results are documented in section IV. We review related work to the problem we study and relevant new contributions in this field in section V, and a brief discussion on our findings is summarized in section VI.

II. System Model

We consider a single-radio single-channel (SRSC) wireless network[1] represented by a directed graph $G(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ denote the sets of nodes and links, respectively. A communication link exists from node u to node v if node v is within the *communication range* of node u, which is denoted by the tuple (u, v). Each link has a physical transmission capacity $c(u, v)$, specifying the peak data rate this link is able to support.

We adopt the protocol interference model [15] to characterize the interference relationship in the wireless network. Under this model, the receiver of a specific link can successfully decode the transmitting signal if it falls outside the *interference range* of the transmitters of other activated links. In other words, links that interfere with each cannot be activated simultaneously. Besides, a node can serve at most one active link at a given moment. The conflict relations among all the links can be represented by a conflict graph [1]. Each node on the conflict graph represents a link on G, and two nodes are adjacent if and only if these two links cannot transmit simultaneously. In this way, links can be scheduled to transmit only if they form an independent set (IS) on the conflict graph.

Regarding the MCF problem, there is a set $\mathcal{K}$ of node pairs where a source node needs to transmit to a destination node, by using multiple hops of wireless links. Each element in

[1] Note that a generic multi-radio multi-channel (MRMC) wireless network can be mapped as a virtual SRSC with the multidimensional tuple modeling technique developed in [11].

commodity flow demands $\mathcal{K}$ is identified by its source and destination node id, denoted by b_k and d_k, respectively. We consider an MCF problem with concerns of both throughput and fairness. More precisely, the objective of the studied problem is to maximize the minimum commodity flow in the network. In a wireless setting, due to link interference, this network flow problem involves not only routing decisions but also scheduling decisions in which ISs are activated in a time-multiplexing manner.

Formally, let $f_k(u,v)$ denote the amount of traffic flow associated with commodity $k = 0, 1, \cdots, |\mathcal{K}| - 1$ on the link (u, v). Besides, let $\mathcal{M}$ be the set of all ISs and α_m denotes the fraction of time allocated to IS m. For commodity k, the achievable throughput r_k is the net flow out of the source node, i.e.,

$$R_k = \sum_{v \in \mathcal{N}^+_{b_k}} f_k(b_k, v) = \sum_{v \in \mathcal{N}^-_{b_k}} f_k(v, b_k), \tag{1}$$

where $\mathcal{N}^-_v$ and $\mathcal{N}^+_v$ denote the set of links going in and out of v, respectively.

We define $p_m(u,v)$ as the effective capacity in an IS m. $p_m(u,v)$ equals to $c(u,v)$ if link (u,v) is activated in m and 0 otherwise. The optimization problem can be formulated as follows.

$$\underset{\{f_k(u,v)\},\{a_m\},z}{\text{Maximize}} \quad z \tag{2}$$

$$\text{s.t.} \quad z \le R_k, \quad \forall k \tag{2a}$$

$$\sum_{k=1}^{K} f_k(u,v) \le \sum_{m \in \mathcal{M}} \alpha_m p_m(u,v), \quad \forall (u,v) \in \mathcal{E} \tag{2b}$$

$$\sum_{u \in \mathcal{N}^-_v} f_k(u,v) = \sum_{u \in \mathcal{N}^+_v} f_k(v,u), \quad \forall v \ne b_k, d_k \tag{2c}$$

$$\sum_{m \in \mathcal{M}} \alpha_m = 1 \tag{2d}$$

$$f_k(u,v) \ge 0, \quad \forall (u,v) \in \mathcal{E}, k \tag{2e}$$

$$\alpha_m \ge 0, \quad \forall m \in \mathcal{M} \tag{2f}$$

In this formulation, z is the minimum of the achieved commodity flow when the problem is solved, guaranteed by the Constraint (2a) and that the problem is a maximization. Constraints (2b) indicate that for any link, the total amount of network flow should not exceed its capacity over a unit time, Constraints (2c) are the flow conservation constraints: for any commodity flow k, and each node v that is not a source or destination node in any demand, the amount of flow entering should equal to that of the flow exiting the node. Constraint (2d) requires that the time fraction assigned to all ISs must sum to unity.

Problem (2) is an LP problem because both the objective and constraints are linear functions of the optimization variables. However, the size of $\mathcal{M}$ is exponentially large and cannot be easily enumerated; even to obtain one set of non-interfering links is equivalent to finding a graph coloring of edges. Therefore solving the problem requires solving a series of integer programming instances followed by an LP with an exponential number of variables. To the best of our knowledge, the most efficient approximation with guaranteed bound analysis is the delayed column generation (DCG) method [14] [2]. The DCG method is based on the observation that, although the size of $\mathcal{M}$ is exponentially large, only a small portion of ISs will be used, i.e., have positive scheduling time, in the optimal solution. Therefore, instead of enumerating all the ISs, the DCG method aims to figure out a set of ISs with a reasonable size in which all these critical ISs are included. With such a IS set, the MCF problem can be readily addressed via an LP solver.

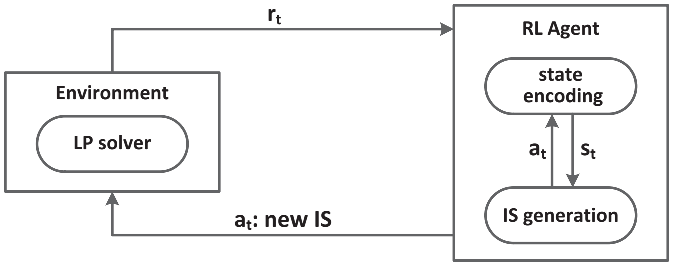


Fig. 1: High-level overview

The DCG algorithm starts with an initial set of ISs that makes the problem feasible and iteratively adds new ISs into that set. At each iteration, an LP problem is firstly solved. This problem, called restricted master problem (RMP), has the same structure of problem (2) except that a subset of $\mathcal{M}$ is considered. The dual information is then leveraged to generate new ISs that have the potential to improve the objective value of the RMP. Using the dual as the link weight, the DCG algorithm selects the maximum weighted link sets that do not interfere with each other as a new set to be added into the current solution. The process of choosing such a link set is termed as solving the sub-problem, and it is an equivalent to solving a maximum weighted independent set (MWIS) problem [2] which is computationally challenging. According to the theoretical results in [14], the solution to the original problem can always be improved in this way until the optimal one is obtained. Additionally, it is proven in [2] that a solution with guaranteed performance bound can be obtained if each sub-problem is solved approximately. Hence, the algorithm stops either when the solution can no longer be improved or upon a predefined maximum iteration number has been reached.

III. Proposed Solutions

In this section we present our proposed solution with a reinforcement learning approach. First we reformulate this problem in the framework of reinforcement learning, and the proposed neural network modules are presented. A high-level overview is shown in fig. 1.

A. Scheduling links as a sequential decision-making

We are inspired by the DCG algorithmic process because each iteration of the algorithm can be seen as a trial-and-error process, in the sense that the solver generates a new column, i.e., an IS of the conflict graph, and receives feedback information from the environment as part of the input for making the next decision. This is akin to the reinforcement learning process, with the important difference that the action is generated not from a fixed rule but must be learned from empirical data.

We adopt the terms used in reinforcement learning literature, and frame the problem as a Markov sequential decision process. We define the *agent* to be the network controller that aims to schedule links through experiencing the environment. At a time step indexed with t, the agent observes the current system *state* $s_t \in \mathcal{S}$, and selects an *action* $a_t \in \mathcal{A}_t$, and receives a scalar reward $r_t \in \mathbb{R}$. The decision by the agent derives from a *policy* π, which maps a state to a probability distribution over the actions available at that time. The system state transitions from s_t to s_{t+1} after the action selection by following a distribution $s_{t+1} \sim p(s|s_t, a_t)$, which can potentially be unknown to the agent.

The goal of the agent is to maximize the expected cumulative reward J, which is defined as the sum of rewards over a time period with maximum length T. Since for each policy π, there exists a corresponding cumulative reward value, this is often written in a functional expression $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T-1} \gamma^t r_t]$, with the future discounting factor $0 \leq \gamma \leq 1$. The expectation is both over the state transitions and action selection. In the context of the link scheduling problem, the above components are defined as follows.

a) State: The state of the system can be determined by the problem input, the current (partial) solution, and the proportion of the link capacity that has been used by the current solution. With the former two, the solver's state is completely specified, but we found that adding the latter helps performance improvement. The state transitions when the agent indicates a new link set to be added, and the master problem is solved again to obtain the new solution. Specifically, the state vector contains information regarding: whether a link is part of a traffic demand, the link capacity, the link source and destination node's coordinates, the current set of ISs, how much capacity is left given the current solution, and the flow amount for each commodity.

b) Action: The action a_t for time step t is the subset of network links that should be added into the master problem. It is a discrete $|\mathcal{E}|$-dimensional vector, and each element is either 0 or 1.

c) Reward: The reward signal for one step is set to the difference of the network utility in this and previous time step. In this setting, we let it be the difference of achieved flow $r_t = z^{(t+1)} - z^{(t)}$. This is to make the agent prefer solving the problem with fewer iterations in order to obtain a fast solution process since, with the discounting factor, the future solution improvement is counted less.

B. Solution overview

Even though the problem can be reformulated as a Markov decision process with ease, there are significant technical challenges that have not received ample attention in the literature. We highlight a few of them as follows and introduces our solution to address them.

First, a proper representation of a variable sized set is needed. Most present solutions of neural networks assume that information under processing has a fixed dimension, especially problems whose solutions are not affected by the variability in the network topology [16], [17]. But in this problem, there may be an unspecified number of links or nodes, and there can be an undetermined number of intermediate ISs, each of which containing an unknown amount of links. To overcome this, we designed our model architecture such that it could adapt to variable problem size and represent quantities with unknown dimensions.

Second, the action space is too large to allow directly learning a policy as the probability distribution $p(a_t|s_t)$. Recall that $\mathcal{A}_t$ represents the set of all conflict-free network links, and is an independent set on the conflict graph. There can be an exponential number of such link subsets, and it changes across different problem instances. In comparison, the agent in most reported works only has to select an action out of a few known actions. As a response, we develop a differentiable searching procedure to address this.

Third, more efficient learning is a must. Direct application of reinforcement learning can be sample inefficient and achieving good performance requires a huge amount of interactions with the problem environment. This can become problematic where the environment evaluation is not cheap. For example, suppose each time the agent selects the action, an external routine is called to evaluate the quality of that selection and this time cost could add up to make the training of a reinforcement agent prohibitively expensive. We design a proper encoding scheme and use curriculum training procedure to make the learning process more efficient.

C. Encoding the link sets

Since the environment state must contain the current partial solution consisting of all the ISs found so far, they must be properly encoded in order to be processed by other parts of the neural network. This is important as the input may present different number of nodes and links, and as a result, the possible number of ISs and the number of links in each IS can vary greatly.

Generic neural networks cannot process such set data, where the order does not matter. To require the neural network to deal with input with varying sizes is to enforce some form of *order invariance*, because the output should be a function of each element's attributes only and not dependent on their order of input to the learning agent.

It is suggested that as long as the model architecture has the form of sum decomposition, its output has order invariance.

Theorem 1 ([18])**.** *A function $f(X)$ operating on a set X having elements from a countable universe, is a valid set function, invariant to the permutation of the instances in X, iff it can be decomposed in the form $\rho \sum_{x \in X} \varphi(x)$, for suitable transformations φ and ρ.*

By this reasoning, we adopt an attention mechanism [19] to encode the link vectors in the set that follows the constraints given above. Suppose link i in a given IS M is represented as a vector l_i. It is a concatenation of link capacity, link node positions, and indicator of flow demand, which marks the link if it is part of a flow demand, as well as how much capacity this link still has given the current solution. The IS is encoded as

$$f_{\text{LL2}}(x, y) \triangleq \text{InnerProduct}(\text{MLP}_1(x), \text{MLP}_1(y)) \tag{3}$$

$$\hat{l}_i = \text{MLP}_2(\sum_{j \in M} \frac{f_{\text{LL2}}(l_i, l_j)}{\sum_{j \in M} f_{\text{LL2}}(l_i, l_j)} \cdot l_i) \tag{4}$$

$$LS(M) = \text{MLP}_3(\sum_{i \in M} \hat{l}_i), \tag{5}$$

where f_{LL2} is the pair-wise function that captures the link interactions in the solution process. MLP stands for Multi-layer perceptron, consisting of several cascading neural network linear layers, each followed by a non-linear activation function. These results are transformed as weights that add up the link vectors to form the intermediate link specific vectors $\hat{l}_i$.

They are then added and then passed through an MLP layer for the link set level representation.

D. Producing link-level searching clues

In each iteration, a new IS is discovered, encoded as previously stated, and added to the set of known ISs. The agent has exact knowledge of these encoded ISs and uses them to generate a per-link vector, to be used in searching for a new IS.

We again make use of the attention mechanism. We denote the encoded, known ISs as a matrix M of shape $|\mathcal{M}_t| \times h_M$, where h_M is the dimension determined by MLP_3 in eq. (3). Let L be the set of link vectors. First, each link vector individually performs attention with the current set of ISs:

$$f_{\text{LL3}}(Q, K, V) \triangleq \text{softmax}(\frac{QK^T}{\sqrt{n}})V \tag{6}$$

$$L_{LS} = f(L, \text{MLP}_4(M), M). \tag{7}$$

Next, each row of the matrix L_{LS} is put into a order-invariant structure to obtain one vector that represents the current problem context h_{context}. The per-link result $l_{\text{out},i}$, where i is the link index, is produced by a pairwise operation with this context:

$$h_{\text{context}} = \text{MLP}_5(\sum_{i=0}^{|\mathcal{E}|} L_{LS}[i]) \tag{8}$$

$$l_{\text{out},i} = \text{MLP}_6(l_i, h_{\text{context}}). \tag{9}$$

The last layer of MLP_6 gives a scalar for each link. The purpose of this step is to generate a per-link signal that incorporates the current situation and each link vector's own attributes. The results act as a "clue" for searching for an IS to give a proper action.

E. Differentiable searching

The agent's policy produces a vector for all links in the network which must be eventually transformed into a link set. We call this a "searching" process because of the similarity of the task's goal: from a continuous vector, the output needs to be a certain subset of a known set of elements. For simplicity, we aim to select greedily a subset with the maximum sum of weights produced previously, as long as the links do not interfere with each other. But this process is not by itself made up of differentiable neural network operations, so back-propagation cannot be directly applied for training.

This is due to the discrete nature of this process. Assuming the input of the search is a continuous variable p and the output z is a discrete variable from a finite set. It can be noticed that the input p's change may not cause a change in z at all. Otherwise, at certain values of p, z change discontinuously. This means the gradient is either 0 or does not exist, causing gradient flow to be useless.

To overcome this drawback, we use an interpolation method [20] which uses a surrogate differentiable function whose gradient information can be used to guide the optimizer for parameter update.

In our case, z is the $|\mathcal{E}|$-dimensional 0-1 vector that forms an IS on the conflict graph, and p is the continuous searching clue l_{out}. As z performs the search for independent set, the relation

$$z(p) = \phi_{z \in \mathcal{Z}}[-z^T p] \tag{10}$$

holds. We consider an continuous approximation that incorporates the final training loss function ξ:

$$z_\lambda(p) \triangleq \phi_{z \in \mathcal{Z}}[-z^T p + \lambda \xi(z)] \tag{11}$$

$$\xi_\lambda(p) \triangleq \xi(z_\lambda(p)) - \frac{1}{\lambda}[-z^T p - z_\lambda^T p], \tag{12}$$

where ξ is the global loss function and λ is a scalar hyper-parameter set experimentally at 20. The gradient we would like to obtain was $\nabla_p \xi$, and is now replaced by the approximate, but more smooth version $\nabla_p \xi_\lambda(p)$.

Differentiating with respect to p gives the correct update formula:

$$\nabla_p \xi_\lambda(p) = -\frac{1}{\lambda}[-z(p) + z_\lambda(p)] \tag{13}$$

$$z_\lambda(p) = \phi[p + \lambda \xi\prime(p)] \tag{14}$$

As a result, the approximate gradient calculated at p is given by

$$\nabla_p \xi \approx \frac{1}{\lambda}(\phi(p) - \phi(p + \lambda \cdot \nabla_z \xi(\phi(p))). \tag{15}$$

The training process itself is a standard policy gradient technique, with the samples sorted according to the difficulty.

IV. Numerical Experiments

In this section, we evaluate the performance of our proposed scheme through different kinds of numerical experiments. Specifically, these experiments examine the performance from these aspects:

- Speed. We would like to see if the RL method can help accelerate the solution of the problems, both in terms of the number of iterations and the absolute time.
- Optimality. We test if the RL-based method results in solutions with a higher quality compared with other heuristic methods.
- Generalization. When the agent sees a problem instance of a different size from the ones of a size it has seen in training, we test if the benefits of speed and quality still persist. We also test the cases where the problem instances are generated with a different distribution than the data used in training.

The problem instances under consideration can be classified into these categories: random, grid, office. In each of the scenario, after the nodes are placed, their channel information is treated as only containing a path loss component, signifying that only the scalar link quality information is available. The link capacity is calculated by Shannon channel capacity formula and it is determined by the physical distance between the source and destination node.

- Random instances are generated by setting a cell area of 1000 m by 1000 m square and randomly placing nodes with a minimum distance of 0.5 m. The interference range and communication range are 50 m and 30 m respectively. The nodes transmit with a transmission power 1 mW. The topology obtained this way can be classified as a *random geometric graph* and is widely studied in networking analysis.
- The grid instances represent a situation where the nodes are placed on a regular, rectangular grid. This scenario is typically used in wireless sensor placement.
- The office instances are obtained by using the layouts in an existing study [21]. The instances are augmented by randomly perturbing the node position within the room.

For random and grid cases, we vary the number of nodes to test the performance under different network sizes. We denote instances with at most 15 nodes as *small*, 50-node instances as *medium* and 200-node instances as *large*.

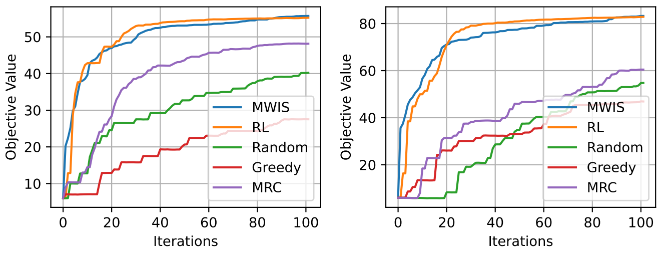
We compare the performance of the proposed method with the following heuristics commonly used in problems of this nature. They follow the framework of iterative column generation, and differ in the way link patterns are chosen. We have chosen this way because the MWIS and Greedy are the most commonly used approaches due to their simplicity; we have not listed other RL methods in network scheduling because they mostly have vastly different network settings (e.g. static wireline networks) to give a meaningful comparison.

- Random. This method selects random links and attempts to pick as many links as possible without causing interference. This amounts to obtaining an unweighted, maximal independent set.
- Greedy. Each link uses the dual variable value provided by the master LP problem as its weight, and builds up a conflict-free set of links with maximal sum weight.
- Maximum weighted independent set (MWIS). Each link also uses the dual variable value provided by the environment, but uses an integer programming solver instead for the true maximum weighted independent set as the next link pattern. The objective value obtained with this method is considered as close to optimum and used as a benchmark for quality comparison.
- Maximum residual capacity (MRC). The links with maximum sum of residual capacity with no conflict are chosen as the next link pattern.

The simulation environment is written in Python programming language, as well as a mixed-integer linear programming solver Gurobi [22] to calculate the state transitions. For neural network and deep reinforcement learning, we use the software frameworks PyTorch [23] and RLLib [24].

A. An example of a typical solution process

As a concrete example, we demonstrate a specific solution process selected from medium and large-sized instances, respectively. In the plots, the horizontal axis is the iteration number and the vertical axis represents the problem specific objective value. As these algorithms explore more link sets, the objective value increases monotonically. The performance of MWIS is treated as a standard baseline, and is seen to be more effective than the other heuristics. We can observe that the proposed method (termed "RL") achieves faster increase in the objective value, often choosing to have larger objective value gains in a few iterations instead of greedily choosing whatever that can improve at each iteration.



(a) A medium instances

(b) A large instance

Fig. 2: Example solution processes for a medium and large instance from random dataset.

B. Time cost

From the examples, it is shown that an RL agent can help identify high quality link sets to help the solution process achieve a close-to-optimum objective value. But a question remains whether the *overall* time cost can be lowered as a result of RL performance. The specific execution time is not an easy figure to measure, because there are many factors affecting whether a fair comparison can be made. For instance, neural networks enjoy highly parallel hardware or software implementations, whose execution time may scale easily with additional computing devices, whereas conventional algorithms rely on the traditional CPU-bound computation that cannot be easily parallelized.

In this subsection, we consider a comparison of the time it takes to solve a problem instance. We do not count training time for RL method, because it is a large but ultimately one-time cost for a deployed model. We define the time cost of RL method as the sum of all action generation and environment transition time, and avoid batching which is not available for traditional algorithms. Similarly, the computation time for other methods is treated as the sum of decision generation time and environment transition time.

	RL	Random	Greedy	MWIS	MRC
random-small	0.60	1.14	**0.52**	0.58	0.55
random-medium	**3.19**	4.94	3.41	3.37	3.43
random-large	**21.55**	-	33.51	28.26	32.10
grid-small	**0.66**	1.58	0.69	0.73	0.95
grid-medium	**2.84**	4.33	3.85	3.45	3.84
grid-large	**20.72**	41.59	38.61	33.80	39.08
office	**1.59**	1.84	2.13	1.74	1.79

TABLE I: Comparison of the average computation time on different types of problem instances. Units are in seconds, and the values are the lower the better. Entries marked with '-' correspond to the situation where a method does not gather sufficient number of cases that reach the target performance within a reasonable time budget.

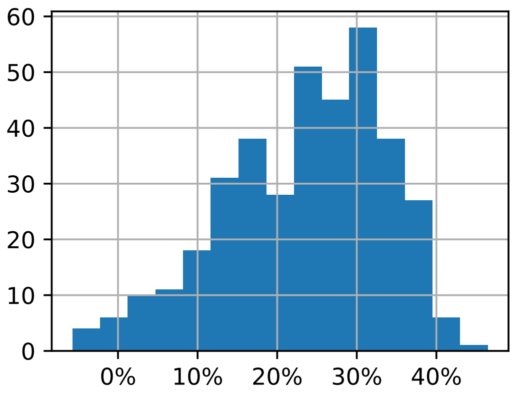


Fig. 3: Histogram of the time savings in random-large datasets. Plotted for the testing data points. The horizontal axis is the time savings percentage number, and the vertical axis is the number of samples

In Table I, we list the run time to reach within 10% of objective value obtained by MWIS algorithm, for different types of problem instances on differently sized networks. We found that even at handicapped position without batching to lower the amortized time cost, on average the RL agent is still time efficient. In Figure 3, the time saving for the random-large samples is plotted in a histogram. We can see that the majority of the time saving is between 15% and 37%, suggesting a significant performance improvement. This is largely due to the similar per iteration action generation time compared with other methods, and also that the number of iterations needed is lower. This effect is more easily observed in larger instances where the proportion of time cost is spent on generating a good link subset.

C. Convergence performance

Since eventually given long enough time, most algorithms can converge to a value close to the optimum, we examine how fast the algorithms reach a target level of performance in terms of iterations. Similarly, we treat the objective value obtained by the traditional algorithm (MWIS) as a goal value, and count the number of iterations it takes to reach that level for all algorithms. In Table II, we show the number of iterations for each type of algorithm on different types of problem instances of different sizes. We can observe from the performance of RL agent that the number of iterations is significantly reduced in larger instances, up to 18%. The lower variance of the results also suggests that RL policy achieves a consistent advantage.

D. Solution optimality

In this experiment, we focus on the large instances to see if, given the same number of iterations, the RL policy reaches a better problem objective value. This is to compare the quality of solution under the same iteration budget. This is different from the experiment in section IV-C, since here the performance improvement within the first few iterations matter more than others. We give a summary of performance comparison for large instances in Table III. When each is given a maximum number of iterations, we see that the RL agent in most scenarios obtains a better objective value. The advantage of the RL policy is clear from the start. While other algorithms suffer from uninformative exploration, the RL agent is able to come up with performative link sets, which contributes to faster value improvement.

E. Generalization to different data

As the performance of reinforcement learning performance heavily relies on the data it has seen during training, we naturally would like to see how a model trained on one type of problem instance can be applied to another type with a different set of data generation parameters.

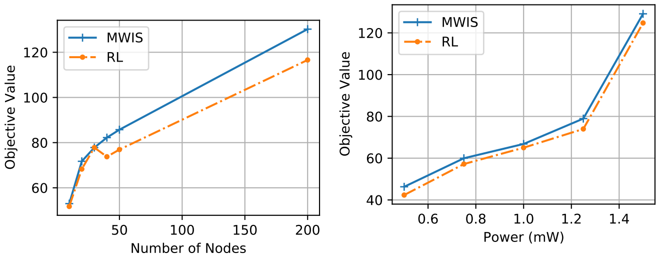
a) Number of Nodes in the Network: We train the agent on instances of smaller sizes and then apply larger instances on testing. The problem size may have an impact on the problem because there are more combinatorial relationships that need modeling. We can observe in Figure 4a that even trained on medium instances, the large instances' performance is still competitive. The performance gap between the differently trained agent exists, but on the whole the performance loss is within 16%. This shows the power of the neural network to learn a good selection criterion that does not simply captures the patterns that are dependent on irrelevant factors like the number of nodes in the graph.

	RL	Random	Greedy	MWIS	MRC
random-small	9.48 ± 10.42	22.77 ± 11.09	16.74 ± 12.31	9.51 ± 7.35	20.15 ± 10.12
random-medium	22.85 ± 14.15	39.85 ± 11.52	42.98 ± 13.84	35.09 ± 16.74	41.59 ± 13.08
random-large	71.55 ± 18.25	-	94.51 ± 26.12	87.50 ± 20.04	96.10 ± 15.39
grid-small	10.75 ± 8.23	19.32 ± 13.60	18.40 ± 16.64	14.33 ± 9.98	17.44 ± 22.39
grid-medium	24.65 ± 13.87	44.62 ± 12.60	42.67 ± 20.11	29.82 ± 17.81	43.49 ± 17.37
grid-large	68.72 ± 16.86	84.37 ± 22.13	88.61 ± 23.46	73.80 ± 22.10	85.35 ± 16.05
office	28.20 ± 12.41	48.82 ± 16.35	48.95 ± 18.37	33.17 ± 16.99	47.47 ± 17.44

TABLE II: Comparison of the number of iterations

	random-large	grid-large	office
RL	69.0 ± 25.2	80.1 ± 25.4	62.4 ± 24.7
Random	54.5 ± 18.4	75.1 ± 17.2	60.0 ± 23.6
Greedy	62.7 ± 19.5	69.7 ± 26.7	58.6 ± 25.0
MWIS	68.4 ± 20.7	80.9 ± 18.3	61.1 ± 21.1
MRC	56.2 ± 23.8	73.7 ± 24.2	59.2 ± 21.3

TABLE III: Objective values achieved by running the algorithm for a fixed number 60 iterations.



(a) Generalization across the number of nodes. Trained on random instances.

(b) Generalization across instance transmission power. Trained on medium size instances with 1.0 mW node power.

Fig. 4: Generalization performance. We train the model on one data point and use the model on data with different parameters and observe the difference in performance.

b) Data generation parameters: Another aspect that may significantly change the data distribution is the generation parameters used. In this experiment, we choose to compare the performance under different transmission power. It is chosen because this parameter directly affects the link quality in the network, therefore causing the interference relationship between the links to vary greatly. In a sense, this parameter has even more impact on the topology formation than the number of nodes. We train the agent on one set of instances with 1.0 mW transmission power and use it on the instances with different power budgets, and compare their performance with the traditional algorithm. In Figure 4b, we observe that the performance is almost on the same level. This suggests that the model's strategy does not change significantly with the data's specific parameter choice and is robust.

V. Related Work

Since link scheduling is one of the most fundamental problems in networking research, numerous works are present to provide theoretical and implementational ideas.

a) Data-driven solutions to network problems: Aspects of network design problems can be cast as optimum control problems, and there have been attempts to apply machine learning methods as a way to discover heuristic algorithms from data. The work in [10] studies the wireless scheduling in a device-to-device (D2D) network and develops a neural network architecture that can judiciously activate a subset of links to yield near-optimal network sum-rate at one shot. [25] typifies the supervised approach to optimize the flow scheduling in wireless ad-hoc networks. Another approach as exemplified in the use of actor-critic style models is reported in [7], where the neural model optimizes the data flow path in a data center network. Similar ideas appear in a series of recent online network control problems [7], [26], [27].

More recently, there is a line of research work which attempts to apply sequence modeling to graphs, with the goal of learning useful problem solutions from learned models. Pointer network [28], a model based on attention multi-head attention mechanism, solves variable-sized combinatorial problems by using the attention scores as selection criteria, and this approach is shown to achieve reasonable performance level with classic problems including Traveling Salesman Problem and Delaunay triangulation. This idea is further expanded to solve a vehicle routing problem [29]. Using the problem graph instance as an input to a transformer, the output is determined by the embedded node vectors. On the outer level, the model is further trained by a policy gradient reinforcement learning algorithm REINFORCE.

Another line seeks a proper representation of the graphical structure in the neural network context. It is shown that by using a structure-aware model `structure2vec` [30], [31], the neural networks can learn to build up approximate solutions by iteratively adding new nodes to an existing partial solution, with the necessary problem-specific helper functions. van Hoof, and Welling 2019a), and the (standard) beam search ver- sion of RL (Nazari et al. 2018). With our beam search scheme, MDAM outperforms not only the sampling version of AM (Kool, van Hoof, and Welling 2019a), but also the improvement heuristic NeuRewriter (Chen and Tian 2019) on CVRP. For the comparison with traditional non-learning based methods, it is worth noting that MDAM outperforms sophisticated general purpose solvers OR Tools 4 and Gurobi (with time limits) on large instances (OP and PCTSP with 100 nodes), and shows relatively good scalability. For some problems such as OP with 50 nodes, MDAM outperforms highly specialized heuristic solvers. For SPCTSP, optimization based methods (e.g. Gurobi, OR Tools and improvement heuristics) require some fo

b) Conventional wireless network optimization: Wireless network optimization had been a key research area in the recent two decades. The basic methodology is to compute the resource allocation aspects such as channel assignment, base station association, scheduling and power control using various mathematical programming algorithms [32]. Due to the complex inference relationship, wireless network optimization is NP-hard in general, and the major thread of efforts in the community is the development of various approximation algorithms [1], [33]. The studies had also been extended from single-radio single-channel context to complex multi-radio multi-channel context [21], [34], [35]. A particular issue inspiring the machine learning study in [25] and this paper is that a new optimization problem instance is always solved either from scratch or with a trivial re-optimization approach [36]; machine learning aims to exploit the historical computation effort to benefit new optimization instances.

VI. Conclusions

In this paper, a deep reinforcement learning based method to the link scheduling problem is presented. The model learns a strategy for choosing link sets iteratively for the overall network flow. To enable flexibility to process variable number of network links and nodes, and allow constrained output, we develop an attention-based neural network module and an augmented module for differentiation. In the numerical experiments, we observe promising performance suggesting that it is able to learn to select link patterns and the benefit is consistent and robust across different configurations.

Acknowledgment

This work was supported in part by the NSF under grants CNS-1816908 and CNS-2008092.

References

[1] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu, "Impact of Interference on Multi-Hop Wireless Network Performance," *Wireless Networks*, vol. 11, no. 4, pp. 471–487, Jul. 1, 2005.

[2] Y. Cheng, X. Cao, X. S. Shen, D. M. Shila, and H. Li, "A systematic study of the delayed column generation method for optimizing wireless networks," in *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing (ACM MobiHoc'14)*, 2014, pp. 23–32.

[3] Y. Cheng, B. Yin, and S. Zhang, "Deep learning for wireless networking: The next frontier," *IEEE Wireless Communications*, to appear.

[4] Z. Qin, H. Ye, G. Y. Li, and B.-H. F. Juang, "Deep learning in physical layer communications," *IEEE Wireless Communications*, vol. 26, no. 2, pp. 93–99, 2019.

[5] H. Huang *et al.*, "Deep learning for physical-layer 5g wireless techniques: Opportunities, challenges and solutions," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 214–222, 2019.

[6] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. of IEEE INFOCOM*, 2018, pp. 1871–1879.

[7] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 191–205.

[8] S. Emara, B. Li, and Y. Chen, "Eagle: Refining congestion control by learning from the experts," in *Proc. IEEE INFOCOM*, 2020, pp. 676–685.

[9] H. Wang, K. Wu, J. Wang, and G. Tang, "Rldish: Edge-assisted qoe optimization of http live streaming with reinforcement learning," in *Proc. of IEEE INFOCOM*, 2020, pp. 706–715.

[10] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1248–1261, 2019.

[11] H. Li, Y. Cheng, C. Zhou, and P. Wan, "Multi-dimensional conflict graph based computing for optimal capacity in mr-mc wireless networks," in *Proceedings of IEEE 30th International Conference on Distributed Computing Systems*, 2010, pp. 774–783.

[12] Y. Cheng, H. Li, D. M. Shila, and X. Cao, "A Systematic Study of Maximal Scheduling Algorithms in Multiradio Multichannel Wireless Networks," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1342–1355, Aug. 2015, ISSN: 1063-6692, 1558-2566.

[13] A. Melchiori and A. Sgalambro, "A matheuristic approach for the Quickest Multicommodity k-Splittable Flow Problem," *Computers & Operations Research*, vol. 92, pp. 111–129, Apr. 1, 2018.

[14] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1997.

[15] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[16] H. Zhang, W. Li, S. Gao, X. Wang, and B. Ye, "Reles: A neural adaptive multipath scheduler based on deep reinforcement learning," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1648–1656.

[17] S. Chinchali *et al.*, "Cellular network traffic scheduling with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[18] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Proceedings of Advances in neural information processing systems*, 2017, pp. 3391–3401.

[19] A. Vaswani *et al.*, "Attention Is All You Need," Jun. 12, 2017. arXiv: 1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762.

[20] M. Vlastelica, A. Paulus, V. Musil, G. Martius, and M. Rolinek, "Differentiation of Blackbox Combinatorial Solvers," presented at the International Conference on Learning Representations, Sep. 25, 2019.

[21] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, 2004, pp. 114–128.

[22] Gurobi Optimization, LLC, *Gurobi optimizer reference manual*, 2020. [Online]. Available: http://www.gurobi.com.

[23] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[24] E. Liang *et al.*, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 3053–3062.

[25] L. Liu, B. Yin, S. Zhang, X. Cao, and Y. Cheng, "Deep learning meets wireless network optimization: Identify critical links," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 167–180, 2020.

[26] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50–56.

[27] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 393–406.

[28] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," in *Proceedings of Advances in neural information processing systems*, 2015, pp. 2692–2700.

[29] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" In *Proceedings of International Conference on Learning Representations (ICLR)*, Feb. 7, 2019.

[30] H. Dai, B. Dai, and L. Song, "Discriminative Embeddings of Latent Variable Models for Structured Data," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 2016, pp. 2702–2711.

[31] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proceedings of*

Advances in Neural Information Processing Systems, 2017, pp. 6348–6358.

[32] Z. Han and K. R. Liu, *Resource allocation for wireless networks: basics, techniques, and applications*. Cambridge university press, 2008.

[33] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends® in Networking*, vol. 1, no. 1, pp. 1–144, 2006.

[34] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2005, pp. 133–144.

[35] L. Liu, Y. Cheng, X. Cao, S. Zhou, Z. Niu, and P. Wang, "Joint optimization of scheduling and power control in wireless networks: Multi-dimensional modeling and decomposition," *IEEE Transactions on Mobile Computing*, vol. 18, no. 7, pp. 1585–1600, 2018.

[36] D. P. Bertsekas, *Network optimization: continuous and discrete models*. Athena Scientific Belmont, 1998.