Experience-driven Wireless D2D network Link Scheduling: A Deep Learning Approach

Shuai Zhang*, Wenlong Shen*, Max Zhang†, Xianghui Cao‡, Yu Cheng*
Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616
AT&T Labs Research, Middletown, NJ 07748
School of Automation, Southeast University, Nanjing, 210018, China

***Abstract*—The protocol design of device-to-device (D2D) networks have regained research interest in recent years, due to the increasing number of networking devices and the diverse deployment settings. Most of the network optimization tasks are fundamentally difficult NP-hard problems in wireless settings, because managing interference introduces combinatorial complexity. Existing approaches use general heuristic algorithms for the underlying graph problems. While efficient and simple, they are not adaptive to the changing requirement and priorities of the service providers, and make no use of the past data to recognize and exploit the information within. In this paper, we study a representative network optimization task of maximizing the throughput-based system utility through link scheduling in a single-radio, single-channel D2D networks, and propose a learning-based method to leverage past experience to generate a good scheduling policy. We combine the pattern matching capabilities provided from recurrent neural networks (RNN) and the flexibility in changing environment from reinforcement learning (RL). The algorithm is implemented with existing software frameworks and tested with numerical experiments. We find that its overall solution quality is comparable to existing heuristics with various network scales, and report an improved system throughput with significant lower computation time.**

***Index Terms*—device-to-device network, deep learning, network utility maximization**

I. Introduction

The next generation communication system will go through a fundamental change in its paradigm. In addition to providing higher data rate to more users, it is envisioned to bring connectivity to heterogeneous devices. In an attempt to increase the flexibility and robustness of network organization, device-to-device (D2D) communication is a promising candidate actively explored in many applications, including internet-of-things, vehicle-to-vehicle networks. Although the upper layers in the networking stack may change their function and abstractions, there are two fundamental limitations that require special attention in deploying wireless D2D networks: 1) the amount of orthogonal resources is increasingly limited, and 2) the user devices' computation capability cannot be presumed to increase indefinitely, due to the impending obsolescence of the Moore's laws and the proliferation of low-cost networked smart devices. As a result, how to efficiently schedule the scare resources with lower overhead to the clients is a critical issue in designing future wireless networks.

In a typical setting, D2D communications take place in Gaussian interference channels, and the network designer is tasked to coordinate the devices' transmissions to maximize system metrics, often a function of system throughput, subject to minimum quality-of-service requirements for the individual links. Through power allocation, transmitters tune their signal power levels to accommodate for the path loss attenuation and prevent jamming their neighbors; In the worst case all transmitters are at their max power with very low data rate achieved.

As a result, the introduction of link scheduling is necessary. By selecting subsets of links to transmit simultaneously, the seriously interfering pairs are designated different time slots to transmit [1]. Since the data rate is a non-linear function of SINR, even if the links must time-share, the improved SINR leads to a significant gain in the combined system throughput.

However, the D2D link scheduling problem presents a significant technical challenge. Although it can be argued that scheduling is a special case of power allocation, with the links not selected set to transmit at zero power, to solve it exactly amounts to solving an integer combinatorial optimization problem, and has been shown to be in the class of NP-hard. Research efforts have been on developing heuristics that give approximate solutions. The majority of them are based on finding independent sets in a conflict graph, which is a representation of the conflict relationship between the links. The exact standards for "conlict" vary: in one work [2], it defines conflict as having their individual SINR lower than a certain threshold if transmitting at the same time; others may define it based on more nodes or the physical proximity. These standards may be otherwise known as the physical or protocol interference models, but in the end the scheduling is reduced to finding a maximal or maximum weighted independent set (MWIS) on the conflict graph. This opens the door for applying the approximation results from the graph theory literature [3].

Another more ad-hoc thread of research aims to directly construct a measure of link importance, usually as a function of the link metrics such as interference and signal strength. The benefits of such approaches are two fold: first, it is easy for building greedy heuristics — start with an empty link set, and iteratively select from the unchosen links a link with the maximum measure, without conflicting too much with chosen ones; second, it is convenient to add additional constraints such as fairness between links or priority by adding penalty or bonus factors in the measure calculation. Recent novel schemes [4]–[6] follow this route to achieve additional benefit.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

In this paper, we are interested in a generalization of the second approach. We explore the latent relationship between link metrics and scheduling decisions through the use of neural-network based machine learning techniques. Today neural network-based learning algorithms have made breakthroughs in many problems traditionally held to be infeasible. While we do not believe that learning algorithms can overcome the complexity of NP-class problems, we consider them to hold promise for network designers based on the following observations. First, the distribution of real-world "best-subset" type of optimization solutions are concentrated in a small region in a large search space. There is often a pattern that can be exploited, but is not currently applied since the current solutions do not make use of past experience to make future decisions. Second, the engineering practice favors adaptability, and machine learning algorithms fill the need, while hand-tailored heuristic rules tend to be designed with specific and fixed assumptions in mind. Network researchers are increasingly more interested in finding out how machine learning could be exploited to help discover ways to optimize the network. Their application in network protocol designs have been explored in many areas such as flow engineering, routing and anomaly detection, where their usage has achieved significant improvement.

Focusing on the making scheduling decisions based on link information, our work combines two algorithmic components from the state-of-the-art artificial neural networks. To the best of our knowledge this is the first work combining these two aspects to solve a wireless D2D network scheduling problem. The first part is *recurrent neural networks* to process sequential data, digesting the sequence of link data to give an output sequence of links to use. The second part, *reinforcement learning*, is an algorithmic framework to derive the best-action policy in a Markov environment to accumulate maximum amount of rewards. It is able to guide the sequence processing part to improve over repeated interactions with the environment, which corresponds to maximizing the system-throughput based utility over time. We show through simulations that our scheme can produce comparable high-quality solutions compared with existing works, and that it achieves such performance with less computation time without the need to hand-craft data features or heuristic metrics.

The rest of the paper is organized as follows: in Section II the mathematical model used for the network is introduced and a basic analysis of the problem complexity is demonstrated; in Section III we start from concepts of machine learning techniques and goes on to explain the effect and workings of each part of the design, and how they match our problem under consideration; in Section IV we detail the numerical experiments to verify the proposed scheme's achieved benefits, with a comparison to the existing algorithms, and we summarize our findings in the the final section.

Notes on mathematical notations: calligraphic letters ($\mathcal{N}$) denote sets while lower-case letters denote single variables. Bold-face letters are used to stress that a variable is a vector or matrix, and the notation $f(x;\boldsymbol{\theta})$ means that the function f takes x as the input and is parameterized by vector $\boldsymbol{\theta}$.

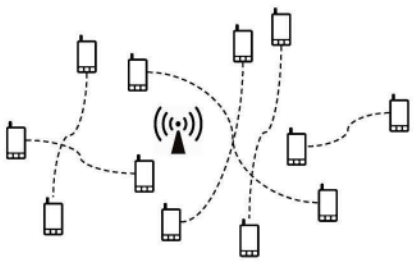

Fig. 1: System illustration

II. Models and Setting

We are interested in the general case of a D2D network, as shown in Fig. 1. A set of communication nodes $\mathcal{N}$ is randomly distributed within a square area. The transmission demands are given in the form of node pairs $\mathcal{L} = \{l_1, l_2, \cdots, l_{|\mathcal{L}|}\}$ and each link is a transmitter and receiver tuple: $l_i = \{t(i), r(i)\}, \forall i \in \{1, 2, \cdots, |\mathcal{L}|\}$. Assume that each node is equipped with a single radio interface, as the multi-radio case can be reduced to the single-radio case by converting the node into parallel single-radio nodes. All transmissions take place using a shared, fixed block of bandwidth W, and the system has time slots which allows time multiplexing. We consider the co-channel interference to the receiving nodes to be the sum of all transmitted signals from other node pairs.

In order to maximize the throughput-based system performance, we assume that there are two decisions to make: *scheduling*, by which a node pair decides if they should communicate at all; *power control*, where the transmitting nodes individually tune the transmitting power to achieve good overall system metrics, through some form of coordination or information sharing.

A commonly adopted received signal model is used in the following discussion. We use $h_{i,j}$ to denote the "out-pair" channel state information (CSI) from the transmitter of link i to the receiver of link j, and $h_{i,i}$ denotes the "in-pair" CSI within the link i. At a receiver node of link l_i, the received signal is the sum of all the other transmitter signal weighted by the CSI and the power level control p_i, plus the Gaussian receiver noise $n_i \sim \mathcal{N}(0, \sigma^2)$:

$$y_i = \sum_{j \in \mathcal{L} \setminus l_i} h_{j,i} \sqrt{p_j} x_j + n_i, \tag{1}$$

where the signal x_j has a pre-determined power $E(|x_j|^2) = P_0$. The effective signal-to-interference-plus-noise ratio (SINR) then can be summarized as $\Gamma_i = \frac{|h_{i,i}|^2 p_i P_0}{\sigma^2 W + \sum_{j \in \mathcal{L} \setminus l_i} |h_{j,i}|^2 p_k P_0}$.

For convenience, we can use the logarithm of the power with the base as P_{ref}:

$$\Gamma_i = \frac{P_{\text{ref}}^{a_{i,i}+b_i}}{\sigma^2 W + \sum_{j \neq i} P_{\text{ref}}^{a_{j,i}+b_j}} \tag{2}$$

$$P_{\text{ref}}^{a_{i,j}} = |h_{i,j}|^2 P_0, \quad P_{\text{ref}}^{b_i} = p_i, \tag{3}$$

where $a_{i,j}$ and b_i are the logarithm of the channel and power control magnitude, and we call the former *effective channel strength*.

We use Shannon's information theoretic formula to calculate the achievable rate per unit of spectrum resource for a node pair:

$$r_i = \log(1 + \Gamma_i), \tag{4}$$

and the achievable rate vector $\mathbf{r} = (r_1, r_2, \cdots, r_{|\mathcal{L}|})$ is defined as the all the rates from the node pairs.

The design goal is to maximize the system utility, which is the long-term average of the instant utility $\mathbf{r}_\tau$: $\lim_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} U(\mathbf{r}_\tau)$. The function U is assumed to be concave and non-decreasing for each component in the input. This assumption ensures that when this measure is maximized, all the components must already be as large as possible; it also has the good property that when the solution space is convex, the optimum exists and is unique. The input of the problem consists of all the effective channel strength $\mathbf{x} = \{\{a_{1,1}, \cdots, a_{|\mathcal{L}|,1}\}, \{a_{1,2}, \cdots, a_{|\mathcal{L}|,2}\}, \{a_{1,|\mathcal{L}|}, \cdots, a_{|\mathcal{L}|,|\mathcal{L}|}\}\}$, and the schedule policy $\pi(\mathbf{x})$ returns a series of link subsets to transmit. We defer the power allocation to a later step by setting all b_i to zero, the optimum scheduling within a time horizon T is formulated as a network utility maximization (NUM) problem in the most general form:

$$\begin{aligned} \text{maximize}_\pi \ & \frac{1}{T}\sum_{\tau=1}^{T} U(\mathbf{r}_\tau) \\ \text{s.t. } & \mathbf{r}_\tau \in \mathcal{R}_\pi(\mathbf{x}) \quad \forall \tau \end{aligned} \tag{5}$$

, where $\mathcal{R}_\pi(\mathbf{x})$ is the space of the achievable rates given the link information $\mathbf{x}$ under a scheduling policy π. After solving this problem, the system power can be further lowered by solving for optimum power allocation vectors, but it is outside the scope of this paper.

III. Machine-learning based Scheduling

The separation of scheduling and power allocation provides a good entry point for the introduction of machine learning techniques: in recent works [7], it has been show that the scheduling decisions often uses a limited number of link subsets. We note that the search for suitable subsets in an exponentially large search space bears remarkable similarity with the recent successful application of reinforcement learning in the field of gaming control [8]. Multi-layer neural networks can be universal approximators given the right amount of data and proper training method; this property could be leveraged for adapting to the traffic pattern in the search of good subsets.

A. Sequence Data Prediction with RNN

In our problem model, all the information needed for scheduling is contained in the effective channel strength between all links. And the scheduling decisions can be conveniently expressed by a vector $\mathbf{y}$ that represents the subset of links to be scheduled. This mapping relationship should be stable against different network configurations: the physical locations of the communication nodes and the channel states are all contained in the effective channel strength vectors, so if a meaningful mapping relationship could be learned, it applies to many network states.

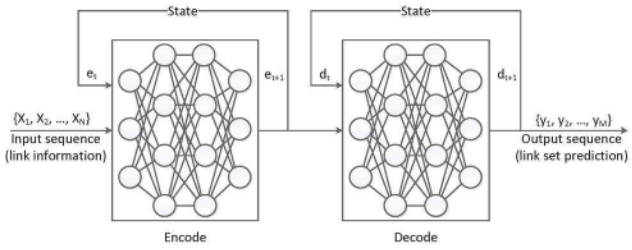

Fig. 2: Sequence Processing. Both the encoder and the decoder are made up of multiple layers of artificial neurons which are non-linear functions. The layers are connected by linear transformations. RNN has its state as an input to change with the incoming new inputs.

Processing such sequential type of data is implemented with the recurrent neural network (RNN) structures widely used in machine translation tasks [9]. The idea of using RNNs is to give an approximate measure of the conditional probability distribution of the output sequence given the input sequence. Specifically, consider that each entry in a dataset is an input and output pair $(\mathbf{x}, \mathbf{y})$, and the input is a sequence of vectors $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$, where all $\mathbf{x}_i$'s vectors of a fixed dimension, and the output is another sequence of potentially different length $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m)$, where $\mathbf{y}_i$'s can have a different dimension than that of $\mathbf{x}_i$.

In this setting the neural network finds the probability through the tuning of its parameter set $\boldsymbol{\theta}$:

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{m} p(\mathbf{y}_i|\mathbf{y}_{i-1}, \cdots, \mathbf{y}_1, \mathbf{x}) \approx f(\mathbf{x}; \boldsymbol{\theta}^\star), \tag{6}$$

where $f(\cdot; \boldsymbol{\theta})$ is a neural network and the first equality derives from the Bayesian chain rule.

The basic structure for sequence processing is illustrated in Fig. 2, with two parts: The *encoder* is an RNN which maintains an internal state that changes as it takes an input $\mathbf{x}_i$. As the entire $\mathbf{x}$ is fed to it, its final internal state will contain information derived from the whole sequence. The *decoder* neural network takes the encoder state as the input, and sequentially produces the conditional probabilities $p(\mathbf{y}_i|\mathbf{y}_{i-1}, \cdots, \mathbf{y}_1, \mathbf{x})$. The $\mathbf{y}$ value with the maximum probability at each step is taken as the inferred item $\mathbf{y}_i$.

B. Reinforcement Learning

Another component of our proposed scheme is reinforcement learning [10]. It is an algorithmic framework for optimizing an agent's rewards in an Markov environment. Compared with the supervised methods where the training is done with existing best control action, we consider the RL approach to have the following advantages in our problem setting: first of all, the system is left on its own to discover a good policy with

performance feedback, and the training can potentially explore better alternatives to known approaches; second, it reduces the requirement of training cases to be provided with high quality target solutions, which can be prohibitively infeasible in many difficult network problems; third, with no preference for one specific optimized target, RL frameworks can be tailored to optimize different system metric.

In the RL terms, at each time slot t, an *agent* is an entity able to perform *actions* $\mathbf{a}(t)$ in a system's *environment*. An environment is assumed to have an internal states $\mathbf{s}(t)$, with Markovian property: the next system state is a function of the current state and action only, independent from the history when conditioned on the current time information. The agent receives a reward based on the environment state and action, and his goal is to maximize his total rewards over time.

Since the agent does not know the exact relationship governing how much reward one can receive given a state and an action, it is necessary to interact with the system for multiple rounds in order to get an idea and improve his *policy* $\pi(\mathbf{a}(t)|\mathbf{s}(t))$, defined as the probability distribution of actions to take when given a certain system state observation (the deterministic policy is a special case). The policy update makes use of the *state-value* function given in Eq. (7), defined as the sum of expected current and future rewards, either a direct sum or a exponentially weighted sum when the systems is at state $\mathbf{s}$ to encourage a trade-off between immediate and long-term benefits. The state-action value function, or *Q-value* function is defined with the action as an additional input, and is connected with value functions through a summation over actions.

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^k R_{t+k}|s] \tag{7}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{k=1}^{\infty} \gamma^k R_{t+k}|s,a] \tag{8}$$

$$V_\pi(s) = \sum_a \mathbb{E}_\pi Q_\pi(s,a)\pi(a|s) \tag{9}$$

The above model is a *model-free* one, meaning that the agent assumes no prior knowledge of the environment or reward, and does not use any special knowledge other than his observations and rewards to devise a policy.

The solution quality of this framework depends on how well the value functions or policy distribution can be learned. As a result there are two approaches, one is to directly optimize the policy itself, and the other is to estimate the Q-value function. And with neural networks, these functions are parameterized, and finding good function approximations is translated to searching for the best coefficients for the network layers.

The current state-of-the-art RL techniques favor a mixture of both, called actor-critic [11]. It consists of two parts: critic network updates the value function parameters, usually in the form of Q-value functions; and the actor network updates the policy parameter according to the information from the critic. In each iteration the critic network predicts the value of the current policy, and is an estimation of the Q-value function for the actor to choose actions from. The actor network improves the policy iteratively, performs a policy gradient descent to find actions based on the critic's values. They are two different neural networks with separate parameters, so in each iteration there are two update operations.

The update of the parameters in the actor and critic networks can be derived as the application of gradient descent and temporal-difference learning:

$$\theta = \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t)(\sum_{t'=t}^{T} r_{t'} - b_t), \tag{10}$$

where α is the learning rate and b_t is the value provided by the critic as a reference.

The above two components form the data-driven scheme to develop a link scheduling mechanism. The policy network and the critic value network are both made up of RNN sequence processing blocks described in Section III-A, but parameterized differently with $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_C$. In the training process, the state is defined as $\mathbf{s}_t = [\mathbf{x}; u_t]$, where $u_t \in Z^{|\mathcal{L}|}$ is vector recording how many times each link has been scheduled so far; the state has Markovian properties, since it depends only on the current state and action. The action $\mathbf{a}_t$ is the vector of link subsets, and the reward r_t is the system utility $U(\mathbf{r}_t)$. We train the networks with a batched process: in each iteration, we generate B cases, and for each case we run the network for a fixed number of time slots, calculating the system utility and update the coefficients as needed. Since the parts listed so far are made from differentiable blocks, it can be trained with gradient descent methods. The training algorithm is listed in Algorithm 1.

Algorithm 1: Learning-Based Scheduler Training

```
Train(S, I, B)      // the training set, maximum
 training iterations, the batch size
θ ←Pretrain(S)
/* supervised training of the RNN in
   (Section III-A) with gradient descent      */
for i ←(1, ···, I) do
    s_b ∼ GenerateCase()   ∀b ∈ {1, 2, ···, B}
    a_b, r_b ∼ RunPolicy(s_b)   ∀b ∈ {1, 2, ···, B}
    β_b ←Critic(s_i|θ_c)
    θ_A ←θ_A + α (1/B) Σ_{b=1}^{B} (r_b − β_b) ∇_{θ_A} log p_{θ_A}(a_i|s_i)
    /* gradient descent update. uses the critic
       value and current reward                 */
    θ_c ←θ_c + α (1/B) Σ_{b=1}^{B} ∇_{θ_c} ||β_b − r_b(a_b)||^2
    /* gradient descent update for critic
       network. uses the distance with reward */
end
return θ_A, θ_c
```

C. Data Ordering

An issue related to using RNN is that the result is dependent on the data ordering; In the experiment we found that the input link data ordering has a subtle effect on the final output. It is due to the fact that the encoder itself is a highly-nonlinear

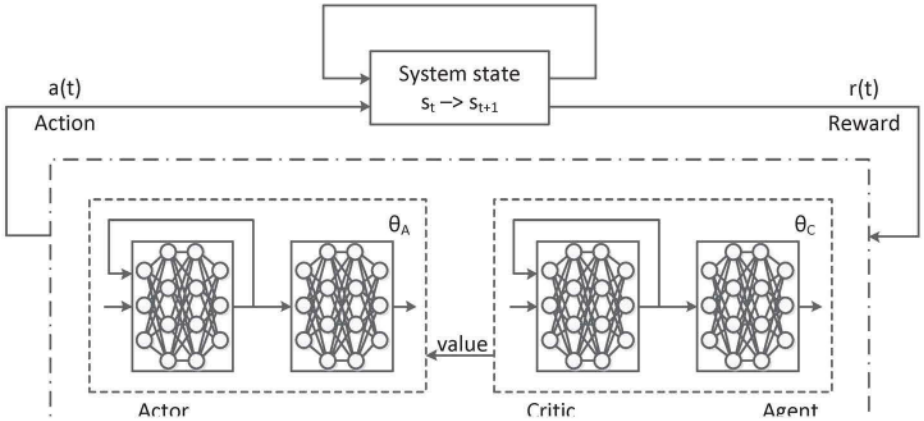

Fig. 3: Diagram of the reinforcement learning.

function, and compositions of them are usually not equivalent; the final encoded state contains information on how the data is ordered. This is useful for data where information order must be differentiated, but undesirable in our case as it introduces unnecessary order dependence and can potentially overfit.

To mitigate such an effect, an additional processing step [12] can be added to mitigate this, with similar forms to the attention mechanism. Instead of passing the new input and the current state (which only contains parts of the input seen so far) to a non-linear function, this step mixes all inputs at each step:

$$\mathbf{w}_t = \text{softmax}(f(\mathbf{x}_i, \mathbf{e}_t)) \quad \forall i$$
$$\mathbf{e}_t = [\mathbf{e}_t, \sum_i \mathbf{w}_t^T \mathbf{x}_i]$$

where $\mathbf{e}_t$ is the encoder's internal state, and function f is a neural network with its own set of parameters, and this function is applied for all $\mathbf{x}_i$ to get a vector of the same length as $\mathbf{x}$. And the final encoder state is obtained after repeating this operation P times, which is part of the hyperparameters to be determined before training. From the expression of $\mathbf{e}$ one can see that it is invariant under the permutations of $\mathbf{x}_i$ because the all the inputs are combined with a weight $\mathbf{w}_i$. This effectively removes the training dependence on the input data order.

IV. Numerical Results

We implement this system with the existing software framework Tensorflow in Python, and conducted experiments to test its system performance. We consider the testing network to be located within a square area of 1000 meters sides, with its illustration shown in Fig. 1 and the parameters listed in Table I. The communication nodes are randomly distributed with a minimum separation of 0.5m to 30m. Although the links are formed by randomly chosen pairs, there is a maximum cap on the distance between the two nodes. This is to prevent the cases where the effective link strength is too weak to make feasible transmission.

We generate 500 configurations in this manner with the total amount of nodes not exceeding 200, then randomly change the link strength for more variations. The amount of training cases total to 20000, and the system utility function is chosen to be an unweighted sum of all link throughput. For comparison purposes, we run two heuristic algorithms that are reported to be efficient for such kind of problems. One is based on the greedy approximation of independent set on conflict graphs [2] to get a baseline comparison; the other is the link SINR based FlashLinQ scheme [13]. The experiments are done on a workstation with a moderate computational power, using Intel i7-6700 processor and 16GB RAM, and the model training part is delegated on an Amazon Web Services p3.2xlarge GPU compute instance.

Carrier Frequency	2.4 GHz
Bandwidth	10 MHz
Maximum Power	20 dbm
Receiver Noise Spectral Density	-173 dBm/Hz
Encoder layers	[256, 256, 256]
Batch size	64
Pair distance	[0.5, 30]
Train, validation, test	18000, 1000, 1000

TABLE I: List of Parameters Used

When the data is used in the learning-based system, we divide them into separate training, validation and test sets. Care has been taken to ensure that only the training dataset is used in the learning process for tweaking the parameters, and the validation data is used for evaluating the training process and hyperparameter setting, not in the learning process. The final reported measures are based on the test data, which is the part the system has never seen in the training, and is used to check whether the system overfits.

In Fig. 4 we show how our algorithms achieve good system throughput when we vary the number of system links in the test cases and plot their average. A high overall throughput is observed, and the gain over the greedy heuristic is consistent across different number of system links, with close to 50% increases in the large system regime. Compared with the other scheme, the performance is on a similar level; the

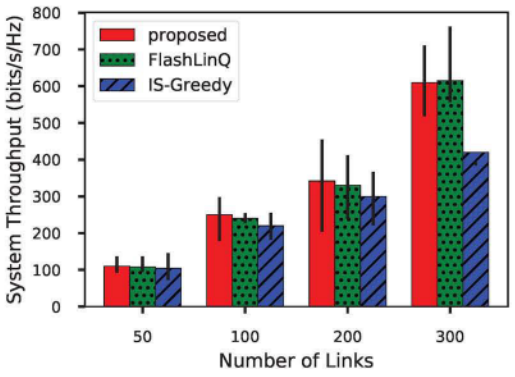

Fig. 4: The average system throughput with the error bar showing the range.

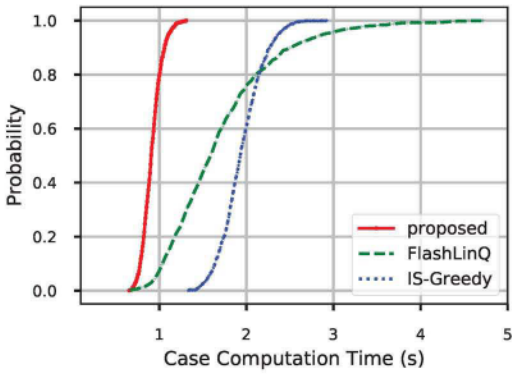

Fig. 5: The cumulative distribution function for the system throughput, compared with existing heuristics based schemes. Graph more to the left is better.

gain in performance is not pronounced, possibly due to its performance close to optimal in the test cases. Despite this, the findings indicate that the machine learning-based scheme could learn a reasonably good strategy. However, the downside is reflected on the vertical bar representing the error range. The solution variance of obtained with the proposed scheme can be significant. This has been an observed issue in many reinforcement learning based methods and it needs to be addressed with additional variance reduction methods in future work.

Next we show a summary of the cumulative distribution function of the average running time to achieve such a performance in Fig. 5. For the proposed learning method we measure the time of making model inference, that is, the step for the *trained* model to make decisions, and the other methods are measured when the main iteration starts to exclude the effects of data loading. Thanks to the fact that the inference task can be massively parallelized onto GPU devices, the learning-based scheme generally works very fast. The results suggest that there is a clear advantage of computation time, with a median computation time reduced by 57.3% and 41.6% respectively.

V. Summary

We propose a learning-based algorithmic framework for generating the link scheduling policy in a D2D networking scenario. The problem is modeled as a sequence data prediction, and combined with the actor-critic reinforcement learning to improve the robustness of the scheduling policy. The findings suggest that it is feasible to discover complex network optimization policies with comparable performance to some existing approaches and low computation cost. The algorithmic components of learning systems can find wide application in network protocol designs, and we expect more work to further improve their efficacy in the future.

Acknowledgment

This work was supported in part by the NSF of USA under Grants CNS-1816908, ECCS-1610874 and the National Natural Science Foundation of China under Grant 61573103.

References

[1] H. Li, Y. Cheng, C. Zhou, and P. Wan, "Multi-dimensional Conflict Graph Based Computing for Optimal Capacity in MR-MC Wireless Networks," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, Jun. 2010, pp. 774–783.

[2] L. Liu, X. Cao, Y. Cheng, L. Du, W. Song, and Y. Wang, "Energy-efficient capacity optimization in wireless networks," in *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 1384–1392.

[3] L. Liu, Y. Cheng, X. Cao, S. Zhou, and Z. Niu, "Joint Optimization of Scheduling and Power Control in Wireless Network: Multi-Dimensional Modeling and Decomposition," *arXiv preprint arXiv:1701.06502*, 2017.

[4] W. Wang, Y. Wang, X.-Y. Li, W.-Z. Song, and O. Frieder, "Efficient interference-aware TDMA link scheduling for static wireless networks," in *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '06, New York, NY, USA: ACM, 2006, pp. 262–273.

[5] B. Hajek and G. Sasaki, "Link scheduling in polynomial time," *IEEE Transactions on Information Theory*, vol. 34, no. 5, pp. 910–917, Sep. 1988.

[6] X. Wu, R. Srikant, and J. R. Perkins, "Scheduling efficiency of distributed greedy scheduling algorithms in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 6, pp. 595–605, Jun. 2007.

[7] 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*, pp. 1–1, 2018.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," p. 9,

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," p. 9,

[10] Y. Li, "Deep reinforcement learning: An overview," Jan. 25, 2017.

[11] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, Progress in Modeling, Theory, and Application of Computational Intelligenc, vol. 71, no. 7, pp. 1180–1190, Mar. 1, 2008.

[12] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv:1506.03134 [cs, stat]*, Jun. 9, 2015. arXiv: 1506.03134.

[13] X. Wu, S. Tavildar, S. Shakkottai, T. Richardson, J. Li, R. Laroia, and A. Jovicic, "FlashLinQ: A synchronous distributed scheduler for peer-to-peer ad hoc networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 4, pp. 1215–1228, Aug. 2013.