Deep Reinforcement Model Selection for Communications Resource Allocation in On-Site Medical Care

Steffen Gracla, Edgar Beck, Carsten Bockelmann and Armin Dekorsy
Dept. of Communications Engineering, University of Bremen, Bremen, Germany
Email: {gracla, beck, bockelmann, dekorsy}@ant.uni-bremen.de

Abstract

Greater capabilities of mobile communications technology enable the interconnection of on-site medical care at a scale previously unavailable. However, embedding such critical, demanding tasks into the already complex infrastructure of mobile communications has proven challenging. This paper explores a resource allocation scenario where a scheduler must balance mixed performance metrics among connected users. To fulfill this resource allocation task, we present a scheduler that adaptively switches between different model-based scheduling algorithms. We make use of a deep Q-Network (DQN) to learn the benefit of selecting a scheduling paradigm for a given situation, combining advantages from model-driven and data-driven approaches. The resulting ensemble scheduler is able to combine its constituent algorithms to maximize a sum-utility cost function while ensuring performance on designated high-priority users.

Index Terms

Allocation, DQN, adaptive, model-selection, 5G, scheduling

I. Introduction

The optimal allocation of communication resources is part of the perpetual race for better performance in mobile communications. As new technologies such as high resolution video streaming and vehicular communications emerge, the demands to be fulfilled by a scheduler are becoming increasingly heterogeneous, resulting in complex optimization tasks that must be solved in real time.

In recent years, deep learning (DL) methods have distinguished themselves with success in complex tasks, including image classification [1] and control [2], among others. DL, as a subset of machine learning (ML), follows a different paradigm compared to classic model-based approaches. Instead of designing an optimal algorithm for a given goal, an algorithm is iteratively approximated based on training data. Without the need to explicitly model the

This work was partly funded by the German Ministry of Education and Research (BMBF) under grant 16KIS1028 (MOMENTUM).
This work was accepted for presentation at IEEE WCNC2022.

underlying processes, the issue of modeling complexity is sidestepped. This characteristic has sparked an interest in research that applies DL to the highly performance-driven field of communication systems.

In particular, increasing the performance and capabilities of mobile communication systems creates new prospects in emergency patient care. For example, video, vitals, or specialist input can be exchanged wirelessly between on-site medical professionals and hospital staff, yielding a significant head start in time-sensitive treatments [3]. Nevertheless, a deliberate approach is required due to the low tolerance for delays and outliers in medical tasks.

Various forays have been made into integrating DL methods with resource allocation tasks, mostly based on deep Q-Networks (DQN) [4]–[6] and actor-critic methods [7]. However, model-based approaches have long thrived, thanks to the valuable expert knowledge that shapes their design. Ideally, incorporating this expert knowledge into data-based methods could govern and stabilize the learning process and produce more tractable algorithms [8].

Therefore, we investigate a combined approach that offers some of the advantages of both DL and model-based approaches. For the task of scheduling discrete resources in vehicle-to-base-station communication, we implement a group of simple, model-based scheduling algorithms. We then train a DQN to select the best model-based algorithm in a given situation, optimizing the long term effect on typical performance indicators: time-outs, sum capacity, and packet rate. Particular attention is paid to the performance of Emergency Vehicles (EV) as a priority class of users. In this way, we are able to use data-driven DL to find an approximately optimal solution to a complex scheduling problem while still using explicitly modeled algorithms.

II. Setup & Notation

In the following, we introduce our system model. We describe the communication channel between a user and the base station, the resources available for allocation, and the generation of jobs. We then formulate the resource allocation problem. After these technical specifications, we move on to the DQN scheduler design.

A. Resource Allocation System Model

In our system model, a number of N user vehicles are connected to a base station, moving on a 2D plane in a grid akin to the Manhattan-model of movement [9]. At any discrete time instance t, vehicles n will take a fixed-size step in a direction of movement that is selected randomly. In doing so, the vehicles have a $98\,\%$ chance of re-selecting their prior direction of movement, $0\,\%$ chance of selecting the direction opposite to their prior movement, and uniform chance of selecting a 90° turn left, right, or stopping.

The communication channel between the base station and a user vehicle n is characterized by its power gain

$$h_n[t] = |\tilde{h}_n[t]|^2 \cdot \mathrm{PL}_n[t]. \tag{1}$$

The power gain is known to the base station. For each simulation time step t, fading channel amplitudes $|\tilde{h}_n[t]|$ are randomly selected from a Rayleigh distribution and multiplied with a distance-proportional path loss factor

$$\mathrm{PL}_n[t] = \min\left(1,\ (d_n[t])^{-1}\right), \tag{2}$$

where $d_n[t]$ is a vehicle's distance from the base station. The path loss factor ensures a degree of correlation of the power gain $h_n[t]$ over time. While exponents of -2 or lower are more typically assumed in modeling real-life

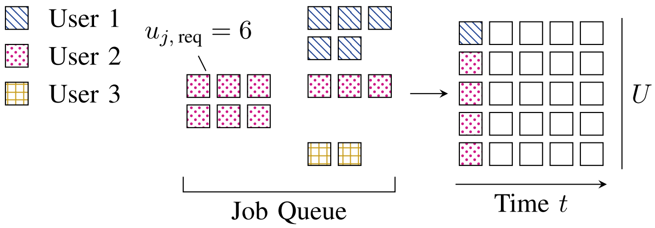


Fig. 1. Jobs consisting of a number of discrete resource blocks arrive in the job queue. A scheduler is tasked with assigning them to a limited number U of resources.

path loss, we selected an exponent of -1 to reduce the spread of the path loss values encountered. For the purpose of this paper, this lessens the computation required in the subsequent learning task.

The base station has access to a limited number U of discrete resource blocks available for allocation, as shown in Fig. 1. These resources can be filled with jobs j from the job queue. At each time step t, new jobs are generated at a probability p_j per user. A job j is assigned to a specific user n and is defined by two attributes, a request size $u_{j,\,\text{req}}[t]$ in discrete resource blocks, and a time-to-timeout $v_j[t]$ in discrete simulation steps. We define the set $\mathbb{J}[t]$ of all jobs in the job queue in time step t and the subset $\mathbb{J}_n[t]$ of the jobs in this queue assigned to user n in t.

The job attributes' initial values are governed by a user profile that declares a maximum job size $u_{n,\,\text{max}}$ and initial time-to-timeout $v_{n,\,\text{init}}$ for users n of that profile. Upon generation, the job is assigned a size $u_{j,\,\text{req}}[t]$ selected from a discrete uniform distribution, $u_{j,\,\text{req}}[t] \sim \mathbb{U}\{1,\, u_{n,\,\text{max}}\}$. When a number of the base station's resources U is allocated to a job j, that job's remaining size is decreased accordingly, while a lifetime count $u_{n,\,\text{sx}}[t]$ of resources scheduled to user n is increased. The jobs' initial time-to-timeout $v_j[t] \leftarrow v_{n,\,\text{init}}$ is decremented for each time step t that passes without the job being fully scheduled. Once the time-to-timeout $v_j[t]$ reaches zero, the job is discarded from the job queue and added to a set $\mathbb{J}_{\text{fail}}[t]$ for that time step t, for use in performance metric calculation.

B. Problem Statement

A scheduler is tasked with assigning the limited number U of resource blocks to the jobs in queue. Three metrics are selected to gauge the performance of a scheduling algorithm:

1) $r_L[t]$: Resource blocks discarded from timeout
2) $r_P[t]$: Global packet rate achieved
3) $r_C[t]$: Global channel capacity achieved

Firstly, the global *sum of resource blocks discarded* due to timing out

$$r_L[t] = \sum_{j \in \mathbb{J}_{\text{fail}}[t]} u_{j,\,\text{req}}[t] \tag{3}$$

should be minimized.

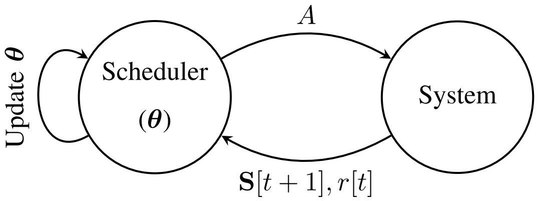


Fig. 2. In the desired RL loop, the parametrized scheduler interacts with the system by selecting an action A and receiving a resulting reward $r[t]$ and new system state $\mathbf{S}[t+1]$. Based on these experiences, the scheduler updates its parametrization $\boldsymbol{\theta}$ to promote high-reward actions and demote low reward actions.

Secondly, a packet rate is introduced as the lifetime ratio of resources requested and scheduled for a user n. We define the set $\mathbb{J}_{n,\,\text{new}}[t]$ as the set of new jobs assigned to user n, generated in time step t. Using the lifetime sum of discrete resources $u_{n,\,\text{sx}}[t]$ scheduled to the jobs of user n, we calculate the *sum packet rate* over all users:

$$r_P[t] = \sum_{n=1}^{N} \frac{u_{n,\,\text{sx}}[t]}{\sum_{\tilde{t}=1}^{t} \sum_{j \in \mathbb{J}_{n,\,\text{new}}[\tilde{t}]} u_{j,\,\text{req}}\left[\tilde{t}\right]} = \sum_{n=1}^{N} k_n[t]. \tag{4}$$

A strong sum packet rate performance is achieved by a scheduler that does not neglect any single user.

Last, the *sum rate capacity* achieved by each transmission is calculated using the Signal-to-Noise ratio (SNR) resulting from the selected channels instantaneous fading characteristics as

$$r_C[t] = \sum_{n=1}^{N} \log\left(1 + h_n[t]\frac{P}{\sigma^2_{\text{noise}}}\right) = \sum_{n=1}^{N} \log\left(1 + \text{SNR}_n[t]\right) \tag{5}$$

for a Gaussian input alphabet, where signal power P and expected noise power σ^2_{noise} are fixed for all vehicles n.

The scheduler is tasked with balancing all performance metrics, thus, we collect all target metrics in a weighted sum utility

$$\tilde{r}[t] = w_C r_C[t] + w_P r_P[t] - w_L r_L[t], \tag{6}$$

with respective tunable weights w_C, w_P, w_L. Additionally, we designate a priority class of EV-type users. Their significance is communicated to the optimization process by adding EV timeouts $r_{L,\,\text{EV}}[t]$ to the weighted sum utility with their own tunable weight $w_{L,\,\text{EV}}$:

$$\begin{aligned} r[t] &= \tilde{r}[t] - w_{L,\,\text{EV}} r_{L,\,\text{EV}}[t] \\ &= w_C r_C[t] + w_P r_P[t] - w_L r_L[t] - w_{L,\,\text{EV}} r_{L,\,\text{EV}}[t]. \end{aligned} \tag{7}$$

III. Algorithm Selection Approach

Where model-based approaches struggle to find an optimal solution, deep Reinforcement Learning (RL) may be applied to learn a function that approximates the optimal algorithm along the domain of reasonable input data. Fig. 2 schematically depicts the desired learning process. At the same time, some applications require boundary conditions to be met, which tend to be easy to formulate in model-based algorithms but cannot be enforced directly in standard RL approaches.

Our scheduler learns to adaptively switch between a selection of model-based algorithms to tackle this problem. In order to select the best algorithm, the scheduler makes use of a deep Q-Network (DQN) [10] to learn the long-term benefit of selecting each operation mode in a given situation. As a result, model-based design benefits, such as hard performance guarantees and human interpretability, are provided by the pool of model-based algorithms. Meanwhile, the superposition of algorithms allows for greater performance than each individual algorithm on flexible goal metrics. As an additional benefit, the individual model-based algorithms do not have to be sophisticated enough to perform well in every circumstance, so long as the overall selection is rich enough to serve any problem.

A. Model-Based Algorithms

Four model-based scheduling algorithms are implemented for the DQN to select from: [11]

(1) A **Maximum Throughput (MT)** scheduler that allocates as many resources as requested by order of descending channel power gains.

(2) A **Max-Min-Fair (MMF)** scheduler looks to distribute the available resources U performantly but fairly among the number $N_{\text{req}}[t]$ of users that have jobs assigned to them in the job queue in time step t. It allocates by order of priority $h_n[t] \,/\, \sum_{j\in\mathbb{J}_n[t]} u_{j,\,\text{req}}[t]$, favoring good channels and small requests, but allocates at most an equal share $\lfloor U/N_{\text{req}}[t]\rfloor$.

(3) A **Delay Sensitive (DS)** scheduler assigns a channel priority

$$p_{c,n}[t] = \frac{k_n[t]}{\sum_{q=1}^{N} k_q[t]} \frac{h_n[t]}{\sum_{q=1}^{N} h_q[t]}$$

given each users relative channel power gain $h_n[t]$ and packet rate $k_n[t]$. The DS scheduler further draws on the users sum timeouts $m_n[t] = \sum_{\tilde{t}=1}^{t} r_{L,n}\left[\tilde{t}\right]$ and lowest remaining time $l_n[t] = \min_{j\in\mathbb{J}_n[t]} v_j[t]$ for a timeout urgency

$$p_{l,n}[t] = \frac{m_n[t]/l_n[t]}{\sum_{q=1}^{N} m_q[t]/l_q[t]}.$$

Each user is allotted a share of the available resources according to the normalized, weighted priority vector

$$\mathbf{p}[t] = (w_1\mathbf{p}_c[t] + w_2\mathbf{p}_l[t])/(w_1 + w_2).$$

Uniquely, this scheduler skips jobs that are about to time out if the allotted discrete resources are not sufficient to complete the job. In this case, the resources are freed for the next highest priority.

(4) An **EV Priority** scheduler assigns as many resources as requested to any EVs and distributes remaining resources one-by-one, randomly assigning them to requesting users.

B. Deep Algorithm Selection

Our deep learning scheduler consists of multiple elements: A pre- and post-processor, a DQN, an $\arg\max$ module, a memory module, and a learning algorithm that tunes the DQN. In this section, we will introduce these modules and illustrate their interconnection.

First, to be able to make informed decisions, a small pre-processor prepares information about the current state of job queue and communication link as a system state vector $\mathbf{S}[t]$. Per user n, the pre-processor summarizes

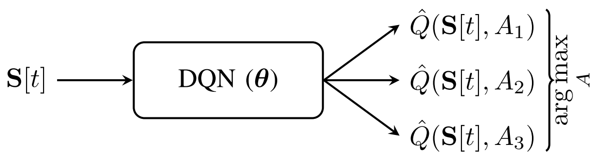

$\mathbf{S}[t]$
DQN ($\boldsymbol{\theta}$)
$\hat{Q}(\mathbf{S}[t], A_1)$
$\hat{Q}(\mathbf{S}[t], A_2)$
$\hat{Q}(\mathbf{S}[t], A_3)$
$\arg\max_A$

Fig. 3. The DQN module estimates expected long term rewards $\hat{Q}(\mathbf{S}[t], A_i)$ for a given state $\mathbf{S}[t]$ and all available actions A_i, according to its current parametrization $\boldsymbol{\theta}$. In this case, actions A_i are the available model-based algorithms. The scheduler then selects the algorithm A_i with the highest expected long term reward.

- current queue length $S_{n,1}[t] = \sum_{j \in \mathbb{J}_n[t]} u_{j,\,\text{req}}[t]$,
- channel power gains $S_{n,2}[t] = h_n[t]$,
- average remaining time $S_{n,3}[t] = \frac{1}{|\mathbb{J}_n[t]|} \sum_{j \in \mathbb{J}_n[t]} v_j[t]$,
- minimum remaining time $S_{n,4}[t] = \min_{j \in \mathbb{J}_n[t]} v_j[t]$,
- and past packet rate $S_{n,5}[t] = k_n[t]$,

for a length of $5N$ features for N users.

For a given system state $\mathbf{S}[t]$ and choice of model-based algorithm A_i, we define the long term expected rewards

$$Q(\mathbf{S}[t], A_i) = \mathrm{E}\left[\sum_{z=t}^{\infty} \lambda^{z-t} r[t] \;\middle|\; \mathbf{S} = \mathbf{S}[t],\, A = A_i \right], \tag{8}$$

with reward r as defined in (7). The long term rewards are discounted by an exponential factor $0 \leq \lambda \leq 1$. As depicted in Fig. 3, a DQN is set up to output an estimate $\hat{Q}$ of the long term expected rewards Q for each choice of model-based algorithm A_i. Given a perfect approximation $\hat{Q} = Q$, we maximize the long term expected rewards by simply selecting whichever action $\arg\max_i \hat{Q}(\mathbf{S}[t],\, A_i)$ has the highest *estimated* long term rewards. Therefore, the learning goal is to update the DQN parameters $\boldsymbol{\theta}$ such that the estimate is approximately close to the true long term reward Q for all possible $(\mathbf{S}, A_i)$, relying on the universal approximation property of neural networks [10].

First, the scheduler must make experiences to learn from. Using an ϵ-greedy exploration scheme, the scheduler initially explores the simulation by taking random actions, i.e., selecting a random model-based algorithm for a given state. States, decisions, and their direct outcome are recorded and stored in a replay buffer in the form of tuples

$$\text{EXP} = (\mathbf{S}[t],\, A[t],\, r[t],\, \mathbf{S}[t+1])\,. \tag{9}$$

We highlight that an experience only contains the immediate reward $r[t]$, not the desired long-term rewards $Q(\mathbf{S}[t], A[t])$, i.e., (8). However, we can extract additional information from $\mathbf{S}[t+1]$, the state that following the action. By again using our DQN, we estimate the rewards following $\mathbf{S}[t+1]$ to construct a learning target

$$\hat{Q}_{\text{target}}(\text{EXP}) = r[t] + \lambda \max_i \hat{Q}(\mathbf{S}[t+1], A_i), \tag{10}$$

that incorporates the information of $r[t]$ and $\mathbf{S}[t+1]$ from the experience. Using this target $\hat{Q}_{\text{target}}(\text{EXP})$, an estimation error [10]

$$\delta = \hat{Q}_{\text{target}}\,(\text{EXP}) - \hat{Q}\,(\text{EXP}) \tag{11}$$

given the networks current parametrization can be calculated for any experience tuple from the buffer.

Following the principle of stochastic gradient descent (SGD), the networks parameters $\boldsymbol{\theta}$ are then adjusted by sampling a mini batch of B experiences from the buffer to minimize a mean square error cost

$$C = \frac{1}{B} \sum_{b=1}^{B} (\delta_b)^2 \tag{12}$$

for the batch. Minibatch parameter updates are carried out every time a new experience is made.

As training progresses and the schedulers understanding of the simulation environment improves, the probability ϵ of selecting random exploration actions is gradually decreased, relying more and more on the network to make decisions. When prompted, at the beginning of a time step t, the network will estimate the expected long term rewards $\hat{Q}$ for selecting any of the available model-based algorithms A_i, given the current state $\mathbf{S}[t]$ and the networks current parameters $\boldsymbol{\theta}$. The scheduler then selects the model-based algorithm A_i with the highest expected long term rewards.

To increase training efficiency, we implement optimizations to the base DQN learning method:

- An infrequently updated network copy is used for the bootstrapping in (10) to increase training stability (*Target Network*, [2])
- Sampling of experiences from the buffer is weighted proportional to the experiences' estimation error magnitude (*Prioritized Replay*, [12])
- The neural network structure is altered to seperately learn the contribution of state and action to the reward estimate (*DuelingDQN*, [12])

IV. Performance Evaluation

A. Implementation Details

We configure the simulation with $U = 16$ available resources and $N = 10$ users. User profiles are set up according to Table I, with five 'Normal', two 'High Datarate', two 'Low Latency' users, and one 'Emergency Vehicle'. For the given configuration, a job creation probability $p_j = 20\,\%$ for each simulation step and user puts a high expected load of 1.6 requests per available resource on average on the schedulers. We run a total of $10\,000$ episodes at 50 time steps t per episode for training and evaluation each, with the mini batch size set to $B = 64$ sampled experiences per training step. Parameter optimization via SGD is carried out by the Adam optimizer [13] with default settings and a learning rate of 1×10^{-4}. Reward weightings are set to $w_C = w_P = 0.25$, $w_L = w_{L,\text{EV}} = 1.0$, giving roughly equal significance to each goal metric with the average expected magnitude of the respective rewards.

For the algorithm selection DQN, a five layer feed-forward network is selected. Layers have 300 nodes each, except for the last layer that was split in two branches with 200 nodes each according to the DuelingDQN structure [12]. Exploration is done by selecting random actions at an initial chance of $\epsilon = 99\,\%$, decaying linearly to $0\,\%$ after $80\,\%$ of training episodes. The exponential future reward decay factor is set to $\lambda = 0.9$ in order to put significance on only a low number of future steps. User TX-SNR $P/\sigma^2_{\text{noise}}$ is fixed to $13\,\text{dB}$.

We implement the simulation in python using the tensorflow library primarily. The full implementation is made available in [14].

TABLE I
USER PROFILES

	Delay in sim. steps	**Max Job Size** $u_{\max}$ in res. blocks
Normal	20	30
High Packet Rate	20	40
Low Latency	2	8
Emergency Vehicle	1	16

B. Results

As the simulation model contains stochastic components, the results achievable on each metric have an inherent variance depending on the specific realizations. For example, a spike in generated jobs will result in timeouts irrespective of the scheduling method. For this reason, results achieved during testing are displayed in cumulative histograms.

Fig. 5 shows each schedulers performance on the combined reward metric r. On this metric, the DQN adaptive scheduler is able to find a strategy that outperforms any single model-based algorithm. Breaking the sum metric r down into its constituent parts sheds light on how this is achieved. As shown in Fig. 4, the DQN scheduler balances the submetrics against each other, yielding some performance on each of them compared to the benchmark.

Of particular interest are the EV-specific timeouts in Fig. 4d. This metric is not specifically targeted by any of the model-based algorithms depicted. A modest double-weighting of EV specific timeouts within the goal metric r, combined with the introduction of the otherwise sub-optimal EV Priority scheduling algorithm to the selection pool, has enabled the DQN based scheduler to significantly suppress timeouts in Emergency Vehicles even compared to the otherwise timeout focused Delay Sensitive algorithm without hurting overall performance overmuch. As Fig. 6 shows, the EV Priority algorithm was only selected a comparably low amount of times to achieve this goal.

The DQNs learned behavior can also be monitored to reveal underlying features of the simulation. As Fig. 6 highlights, for the given reward weighting, the MMF algorithm was only selected a low number of times. Further investigation could reveal whether these are, for example, high impact outlier cases that could be better served with an additional model-based algorithm that is specifically targeted to them, or whether the MMF algorithm is not fit for the given scenario.

The DQN decision making does however add another layer of computation to the scheduling process. Further, while the ensemble method can relax the sophistication required from each model-based part of the ensemble, the composite scheduling function can only assume the function space spanned by the group of model-based algorithms. In other words, the DQN adaptive method is unable to discover strategies that go beyond combining the available models. Determining whether the selection of model-based algorithms provided is rich enough to serve the problem therefore remains a burden on the designer.

V. Conclusion

In this paper we presented a RL-based communications resource scheduler that constructs an effective scheduling paradigm from an ensemble of model-based algorithms. We achieved this by learning to switch to whichever algorithm promises the highest expected long term benefit based on the current queue state. This approach combines the flexible goal optimization of RL methods with the rigid predictability of model-based algorithms. It is noteworthy for applications with complex, conflicting performance goals, where either a) strong models exist to cover parts of the problem, or b) explicit modeling is otherwise necessary, e.g., due to very low tolerance for outliers, such as the transmission of critical medical data. Using this approach unlocks the benefits of RL without abandoning explicit modeling. For the simulation presented, the adaptive model-switching scheduler was able to learn to outperform single, model-based algorithms on a weighted sum utility metric.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Proc. NeurIPS*, vol. 25, pp. 1097–1105, 2012.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] M. Zanatta, P. Benato, and V. Cianci, “Pre-hospital ultrasound: current indications and future perspectives,” *Int. J. Crit. Care. Emerg. Med.*, vol. 2, no. 2, p. 019, 2016.

[4] S. Joseph, R. Misra, and S. Katti, “Towards self-driving radios: Physical-layer control using deep reinforcement learning,” in *Proc. ACM HotMobile*, 2019, pp. 69–74.

[5] H. Ye, G. Y. Li, and B.-H. F. Juang, “Deep reinforcement learning based resource allocation for V2V communications,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3163–3173, 2019.

[6] F. Al-Tam, N. Correia, and J. Rodriguez, “Learn to Schedule (LEASCH): A Deep reinforcement learning approach for radio resource scheduling in the 5G MAC layer,” *IEEE Access*, vol. 8, pp. 108 088–108 101, 2020.

[7] Y. Huang, S. Li, C. Li, Y. T. Hou, and W. Lou, “A deep-reinforcement-learning-based approach to dynamic eMBB/URLLC multiplexing in 5G NR,” *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6439–6456, 2020.

[8] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, “Model-Based Deep Learning,” *arXiv:2012.08405*, 2020.

[9] N. Aschenbruck, E. Gerhards-Padilla, and P. Martini, “A survey on mobility models for performance analysis in tactical mobile networks,” *J. Telecommun. Inf. Technol.*, pp. 54–61, 2008.

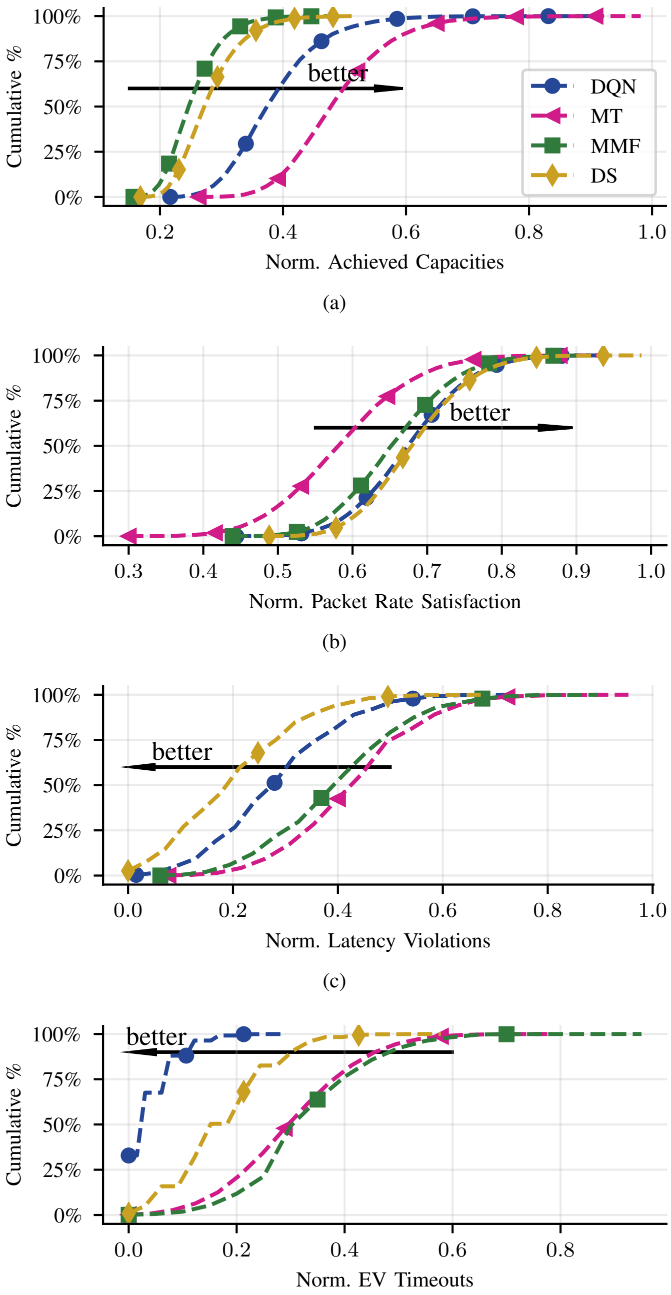
[10] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[11] S. O. Schmidt, “Analyse von Latenz-empfindlichem Scheduling auf Systemebene für Anwendungen der Industrie 4.0,” Master’s thesis, Universität Bremen, Bremen, Germany, 2018.

[12] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in *Proc. AAAI*, vol. 32, no. 1, 2018.

[13] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980*, 2015.

[14] S. Gracla, “Adaptive Scheduling Model Selection,” https://github.com/Steffengra/resourceallocation/tree/master/scheduling, 2020.



(a)

(b)

(c)

(d)

Fig. 4. Cumulative histograms of the schedulers' performance on the individual parts (a) r_C, (b) r_P, (c) r_L and (d) $r_{L,\,\mathrm{EV}}$ of the overall optimization goal r (compare (7)). Dominating all submetrics at once is impossible as the individual submetrics have conflicting objectives, e.g., scheduling an urgent job while the channel is weak. Maximum Throughput (MT), Max-Min-Fair (MMF) and Delay Sensitive (DS) schedulers show strengths in some submetrics but weaknesses in others. The presented DQN scheduler achieves its overall optimization goal by balancing strong performance in all submetrics.

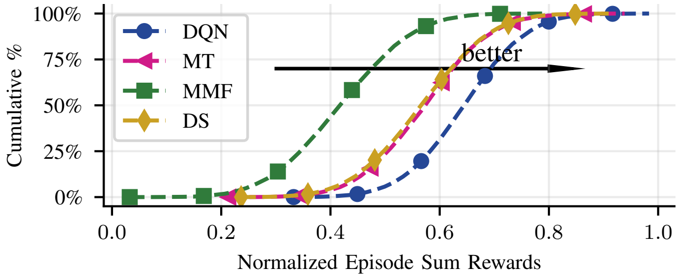


Fig. 5. Cumulative histogram of performance of Maximum Throughput (MT), Max-Min-Fair (MMF) and Delay Sensitive (DS) schedulers as well as DQN adaptive scheduler on the weighted sum reward metric. For each episode, all achieved rewards r are summed. Achieved reward sums are normalized by the highest achieved reward sum.

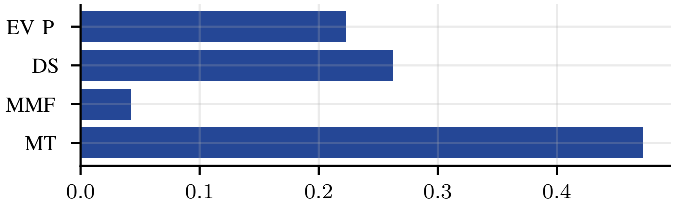


Fig. 6. Relative selection rate of EV Priority (EV P), Delay Sensitive (DS), Max-Min-Fair (MMF) and Maximum Throughput (MT) schedulers during testing.