Inference-based Reinforcement Learning and its Application to Dynamic Resource Allocation

Paschalis Tsiaflakis and Werner Coomans
Nokia Bell Labs, Antwerp, Belgium
paschalis.tsiaflakis@nokia-bell-labs.com, werner.coomans@nokia-bell-labs.com

***Abstract*—Reinforcement learning (RL) is a powerful machine learning technique to learn optimal actions in a control system setup. An important drawback of RL algorithms is the need for balancing exploitation vs exploration. Exploration corresponds to taking randomized actions with the aim to learn from it and make better decisions in the future. However, these exploratory actions result in poor performance, and current RL algorithms have a slow convergence as one can only learn from a single action outcome per iteration. We propose a novel concept of Inference-based RL that is applicable to a specific class of RL problems, and that allows to eliminate the performance impact caused by traditional exploration strategies, thereby making RL performance more consistent and greatly improving the convergence speed. The specific RL problem class is a problem class in which the observation of the outcome of one action can be used to infer the outcome of other actions, without the need to actually perform them. We apply this novel concept to the use case of dynamic resource allocation, and show that the proposed algorithm outperforms existing RL algorithms, yielding a drastic increase in both convergence speed and performance.**

I. Introduction

Reinforcement learning (RL) is a popular machine learning branch that is well established both in academia and industry. Many relevant use cases have been tackled by RL in different research areas, especially in signal processing systems and communication system design [1]. To efficiently solve these RL problems a vast amount of algorithms has been proposed in literature [2], each having its own trade-off in terms of performance, convergence speed and computational complexity. For each problem class, it is important to select the RL algorithm that is most suitable to solve it.

Exploitation and exploration are key mechanisms used in RL algorithms. Exploitation refers to taking actions (in states) that are expected to result in large rewards, whereas exploration refers to taking randomized actions with the aim to learn from it and make better decisions in the future. Striking a good balance between exploitation and exploration is a challenging research topic that is studied elaborately in RL literature [2]. For time-varying environments it is important to configure a high exploration level to be able to track the environment. These exploratory actions often yield a poor performance, which is unacceptable for many applications. Examples of popular exploration strategies [2] are ϵ-greedy action selection, upper-confidence-bound (UCB) based action selection and a soft-max distribution action selection. For dynamic environments adaptive exploration strategies have been proposed in [3].

This work was supported by the Flemisch Government funding agency VLAIO, through the SPIC project (HBC.2020.2197).

This paper proposes another way to resolve the exploration-exploitation dilemma, which can be applied to a particular class of problems, in which the reward and next states of alternative actions can be inferred based on the outcome of a single action. It will be shown that dynamic resource allocation (DRA) for communication systems falls under this proposed class. DRA is important for communication services that have stringent latency and bandwidth efficiency requirements. DRA is employed in multiple recent communication technologies, such as 5G wireless communication [4], the latest DSL technology G.fast [5], [6], as well as time-division-multiplexing Passive Optical Network technologies [7].

The exploration-exploitation dilemma is tackled by a novel concept we call "Inference-based Reinforcement Learning (I-RL)", which exploits the properties of the proposed problem class to eliminate the need for explicit exploration. As a result the convergence speed can be significantly improved, and the performance penalties due to exploration can be avoided.

This paper is organized as follows. In Section II general background is provided on RL algorithms with a concrete explanation of the SARSA algorithm. In Section III the problem class that can benefit from the Inference concept is described. In Section IV the concept of *Inference-based Reinforcement Learning* is proposed and elaborated for the SARSA algorithm, also referred to as I-SARSA. In Section V, I-SARSA is applied to DRA and compared to SARSA.

II. Background on Reinforcement Learning

A Markov Decision Process (MDP) is the mathematical framework that is generally used to model RL problems. An MDP can be represented by a tuple $(\mathcal{S}, \mathcal{A}, R, \mathcal{T})$, in which $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{R}$ denotes the reward space, and T denotes the transition probability function. In a finite MDP, the sets of states, actions, and rewards have a finite number of elements. An RL problem involves an agent and an environment. The agent observes the current state S of the environment, takes an action A that impacts the environment causing it to change its state to the next state S' and enables the observation of a reward $R(S, A)$. From multiple iterations of sequences S, A, R, S' the goal of the RL algorithm within the agent is to derive a policy (i.e.,

a state-to-action mapping) that at each time step k maximizes the expected discounted reward of the environment:

$$\sum_{k'=0}^{\infty} \gamma^{k'} R_{k+k'+1}, \tag{1}$$

where γ is the discount factor, $0 \leq \gamma \leq 1$ [2].

As an illustration of a reinforcement learning algorithm, the popular state-action-reward-state-action (SARSA) algorithm [2] is considered. SARSA will be further extended in Section IV to an I-RL algorithm. Without loss of generality, we focus on a tabular SARSA implementation, as shown in Algorithm 1, from [2], for a setting in which no episodes are used and in which the state and action spaces are discrete. The action-value function $Q(S, A)$ corresponds to a table with the rows corresponding to all possible states and the columns corresponding to all possible actions. Each entry of the table stores the expected (discounted) return for performing the corresponding action A in the corresponding state S, as learned up to that moment by the RL algorithm. Based on the action-value function the best action A' for a given state S' can be determined as

$$A' = \arg\max_a Q(S', a). \tag{2}$$

Algorithm 1 SARSA algorithm

1: Parameters: step size $\alpha \in (0, 1]$, $\gamma \in [0, 1]$, small $\epsilon > 0$
2: Initialize $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily
3: Initialize S and A
4: **while** S is not terminal **do**
5: Take action A
6: Observe reward $R(S, A)$ and next state $S'(S, A)$
7: Choose next action A' for next state $S'(S, A)$ with an exploitation-exploration trade-off (e.g., ϵ-greedy)
8: $Q(S, A) \leftarrow Q(S, A) + \alpha\,[R(S, A) + \gamma Q(S', A') - Q(S, A)]$
9: $S \leftarrow S', A \leftarrow A'$
10: **end while**

SARSA starts with an initialization of the algorithm parameters, the action-value table $Q(s, a)$, the state and action spaces, the state S and the action A. Parameter α is the step size for the gradient-like update formula of the action-state function. Parameter γ is the discount factor that determines the importance of future rewards. Parameter ϵ determines the level of exploration. The main loop starts by taking an action in the current state on line 5 and observing the reward and next state of the environment on line 6. For the next state S', the next action A' is chosen using an exploitation-exploration trade-off depending on parameter ϵ on line 7. A popular approach is the so-called ϵ-greedy selection, where the best next action A' (according to formula (2)) is taken with probability $1 - \epsilon$, and a random next action is taken with probability ϵ. On line 8 the action-value function $Q(S, A)$ is updated for state S and action A. Finally on line 9 the state and action are updated to the next state and next action, respectively.

III. Problem class suitable for Inference-based Reinforcement Learning

Problems considered by RL can have different characteristics ranging from stationary to non-stationary, and from deterministic to stochastic. I-RL is applicable to a specific class of RL problems, that is characterized by the fact that once the outcome (e.g., the reward $R(S, A)$, the next state $S'(S, A)$, or internal environment variables $\vec{\theta}$) of a single action A in a state S is known, one can infer the rewards and next states of alternative actions without actually executing them. This RL problem class will be referred to as the *Inference-capable problem class*. Formally, it can be defined as

$$\begin{aligned} \exists a \in \mathcal{A} \setminus A : \exists f, g : & R(S, a) = f(R(S, A), S'(S, A), \vec{\theta}), \\ & S'(S, a) = g(R(S, A), S'(S, A), \vec{\theta}), \end{aligned} \tag{3}$$

where f and g are known functions (or mappings) that allow to infer the reward and next states, respectively, for alternative actions based on the observation of a single outcome $R(S, A), S'(S, A), \vec{\theta}$. The alternative actions a can refer to all other actions or a subset of actions, the latter corresponding to a setting in which the functions f, g are only known for a subset of actions. For some RL algorithms (e.g., contextual bandit algorithms with discount factor $\gamma = 0$), only the function f needs to exist, as next states are not used in these algorithms.

The Inference-capable problem class (3) covers multiple use cases in different areas, but we will only elaborate it for the DRA use case in Section V.

IV. Inference-based reinforcement learning

Inference-based reinforcement learning is a novel concept for RL that exploits the property of the Inference-capable problem class (3). The main idea is that during a single iteration the outcome (reward and next state) of an action A performed in a state S is used to infer (or estimate) the rewards and next states of alternative actions $a \in \mathcal{A} \backslash A$, which are then used to update the action-value function $Q(S, a)$. Existing RL algorithms only update the action-value function for a single state S and action A pair per iteration. This is visualized in Figure 1, for the case where the set of alternative actions a covers all other actions in the action space $\mathcal{A}$.

This concept is now elaborated for the SARSA algorithm as an example, but it can similarly be applied to alternative algorithms, such as Q-learning, contextual bandit algorithms, etc. The extension of the SARSA algorithm to the *Inference-based SARSA algorithm* (I-SARSA) is shown in Algorithm 2. The initialization steps (lines 1-6) are similar to those of the SARSA algorithm, with the difference that there is no need for the ϵ parameter related to exploration. The difference with SARSA is on line 7, where the rewards $R(S, a)$ for all (or a subset of all) actions $a \in \mathcal{A} \backslash A$ are inferred exploiting property (3). On line 8 the same is done for the next states. On line 9 the next actions $A'(S'(S, a))$ are computed for all possible actions $a \in \mathcal{A}$ in case they would have been selected in the previous iteration. The information obtained in lines 7, 8 and 9 is used

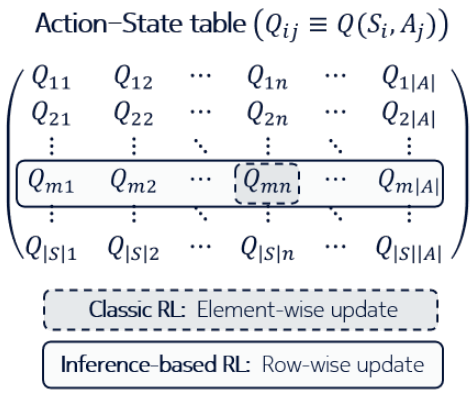

Fig. 1. Action A_n is performed in state S_m. Inference-based reinforcement learning infers the rewards $R(S_m, \cdot)$ and next states $S'(S_m, \cdot)$ of multiple alternative actions $A_1, \ldots, A_{|A|}$ for state S_m based on the outcome of one single action A_n during that iteration. This information is used to update the action-state function $Q(S, A)$ for all entries $Q(S_m, \cdot)$, as opposed to existing tabular RL algorithms that only update a single entry $Q(S_m, A_n)$.

in line 10 to update the action-value function for all possible actions $a \in \mathcal{A}$ in state S, as opposed to only updating the action-value function for a single action A as SARSA does. Finally on line 11 the state is updated to the next state $S'(S, A)$ and the next action is updated to $A'(S'(S, A))$.

The selection of the next action on lines 9 and 11 does not consider any explicit exploration. There is a 100% implicit exploration in the sense that the impact of all actions (for a given state) is continuously inferred, but this does not require actual execution of these actions. As such, there is no performance penalty due to explicit exploration, i.e., no randomized actions with poor performance. This is a desirable property for problems where the performance penalty of randomized actions is unacceptable, or for problems considering time-varying environments for which high exploration is necessary to continuously track changes in the environment.

Algorithm 2 Inference-based SARSA algorithm (I-SARSA)

1: Parameters: step size $\alpha \in (0, 1]$, $\gamma \in [0, 1]$
2: Initialize $Q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily
3: Initialize S, choose $A = \arg\max_a Q(S, a)$
4: **while** S is not terminal **do**
5: Take action A
6: Observe reward $R(S, A)$ and next state $S'(S, A)$
7: Infer rewards for other actions: $\forall a \in \mathcal{A} \backslash A : R(S, a)$
8: Infer next states for other actions: $\forall a \in \mathcal{A} \backslash A : S'(S, a)$
9: Compute next actions for all next states as:
$\forall a \in \mathcal{A} : A'(S'(S, a)) = \arg\max_{\tilde{a}} Q(S'(S, a), \tilde{a})$
10: $\forall \tilde{a} \in \mathcal{A} : Q(S, \tilde{a}) \leftarrow$
$Q(S, \tilde{a}) + \alpha \left[R(S, \tilde{a}) + \gamma Q(S'(S, \tilde{a}), A'(S'(S, \tilde{a}))) - Q(S, \tilde{a}) \right]$
11: $S \leftarrow S'(S, A), A \leftarrow A'(S'(S, A))$
12: **end while**

A rigorous analysis of the convergence speed up of I-SARSA is outside the scope of this paper. The regret is often used as a convergence measure, and corresponds to the expected decrease in reward gained due to executing the learning algorithm instead of the optimal actions from the beginning. In [8], it is proven that Q-learning with UCB exploration achieves a regret $\mathcal{O}(\sqrt{H^3 |\mathcal{A}| |\mathcal{S}| T})$, where H denotes the number of steps per episode, $|\mathcal{A}|$ the number of actions, $|\mathcal{S}|$ the number of states and T the total number of steps. So there is a square root dependence on the action space size $|\mathcal{A}|$. Tabular Q-learning algorithms update the action-value table for a single entry (i.e., state and action combination) per iteration. The proposed I-SARSA algorithm updates the action-value table for all actions of state S per iteration, which increases the number of updates per iteration by a factor equal to the number of states $|\mathcal{A}|$. We conjecture that applying the Inference concept improves the convergence proportional to the factor $\sqrt{|\mathcal{A}|}$. In Section V the convergence speed up of SARSA and I-SARSA will be numerically evaluated for the concrete use case of dynamic resource allocation.

V. Dynamic Resource Allocation by Inference-based Reinforcement Learning

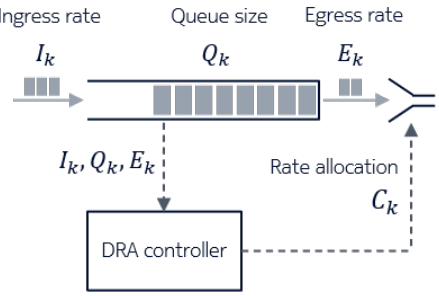

Fig. 2. DRA system model

As shown in Figure 2, we consider a DRA system model consisting of a single (time-slotted) traffic queue or buffer, as follows:

$$Q_{k+1} = \max\left(0, Q_k - E_k + I_k\right), \tag{4}$$

where Q_k and Q_{k+1} denote the queue fill at the start and end of time interval k, respectively, I_k denotes the ingress rate during time interval k, and E_k denotes the egress rate during time interval k. The egress rate is bounded by the allocated resources C_k, under control of the DRA controller, as follows

$$E_k = \min(Q_k + I_k, C_k).$$

To map DRA to an Inference-based RL problem, we consider the system parameters available at the beginning of time interval k, such as the ingress rates during the past 4 time intervals $I_{k-4}, I_{k-3}, I_{k-2}, I_{k-1}$. The goal is to optimize the resources C_k allocated during time interval k so as to maximize the expected discounted reward (1), where the reward is defined as a weighted trade-off between the queue fill (resulting in queuing latency) and the bandwidth inefficiency:

$$R(S, A) = -\max\left(0, Q_k - E_k + I_k\right) - \beta \max\left(0, C_k - E_k\right),$$

where β allows to trade-off the first term relating to latency, and the second term relating to bandwidth inefficiency.

To avoid a very large state space and thus action-state table, we encode the states and actions. Note that for practical implementations on state-of-the-art telecommunications products, it is crucial to target a very low complex solution with small memory footprint. We discretize the ingress rate of the last time interval I_{k-1} uniformly between 0 and a maximum value considered for the ingress rate I^M, with D_S discretization levels. The discretized ingress rate will be referred to as I^D_{k-1}. In addition, the trend of the ingress rate over the four last intervals $I_{k-4}, I_{k-3}, I_{k-2}, I_{k-1}$, is captured by a normalized correlation with linearly increasing values $\alpha = [0.4, 0.8, 1.2, 1.6]$, as follows:

$$c_k^{\text{norm}} = \frac{\sum_{i=1}^{4} \alpha_i I_{k-i}}{\sum_{i=1}^{4} I_{k-i}}.$$

This normalized correlation value c_k^{norm} is encoded with three values, as follows,

$$c^{\text{enc}} = \begin{cases} 0 & \text{if } c_k^{\text{norm}} < 0.95, \\ 1 & \text{if } 0.95 \leq c_k^{\text{norm}} < 1.05, \\ 2 & \text{if } 1.05 \leq c_k^{\text{norm}}. \end{cases}$$

This three-value parameter is also added to the state space encoding, resulting in a total number of states equal to $3 \times D_S$.

The action corresponds to the bound on the egress rate, i.e., C_k, which is also discretized between 0 and I^M, with a number of levels D_A, denoted by C_k^D. The action, state and reward function for time interval k can be mapped as follows:

$$\begin{cases} A & = & C_k^D, \\ S & = & I_{k-1}^D + c^{\text{enc}} D_S, \\ R(S,A) & = & -Q_{k+1} - \beta \max\left(0, C_k^D - E_k^D\right), \end{cases}$$

with $E_k^D = \min(Q_k + I_k, C_k^D)$. The reward observed at the end of time interval k is defined by the trade-off between the queue fill and the unused bandwidth. The number of actions equals D_A, the number of states equals $3 \times D_S$, and the action-state table has $3 \times D_S$ rows, and D_A columns.

One time slot $[k-1, k[$ corresponds to one iteration of the RL algorithm. The queue fill information Q_k, Q_{k+1} and the ingress rate I_k are consulted at the end of time interval k. Knowing Q_k and I_k, the inference property (3) holds. More specifically, the reward for other actions C than that of the performed action C_k^D, can be derived based on the measures I_k, Q_k as follows,

$$\forall C \in \mathcal{A} \setminus C_k^D : R(S, C) = \\ -\max(0, Q_k + I_k - E_k^D(C)) - \beta \max\left(0, C - E_k^D(C)\right),$$

where the set of actions $\mathcal{A}$ corresponds to all possible values of C_k^D, and with $E_k^D(C) = \min(Q_k + I_k, C)$. With these definitions, the I-SARSA algorithm can be applied to minimize the queuing latency and bandwidth inefficiency.

VI. Simulations

To compare the performance of SARSA and I-SARSA for DRA, we consider two traffic patterns: a noisy sine time function (as a toy example), and a real large file download TCP traffic trace captured with a state-of-the-art DSL modem.

The parameter settings for SARSA and I-SARSA are $\alpha = 0.05$, $\gamma = 0.8$ and $\beta = 0.4$. These values are specifically optimized for SARSA to have best performance for the considered traffic patterns. Further, $\epsilon_{\text{base}} = 0.3$ is considered for SARSA. An action-state table with 33 states ($3 \times D_S = 3 \times 11$) and 11 actions ($D_A = 11$) is considered. The action-state table is initialized with zeros. We implemented a minor modification in SARSA that decreases the value of ϵ after 1000 iterations according to

$$\epsilon = \frac{\epsilon_{\text{base}}}{\max(1, 0.001 * \text{iteration})}.$$

The value of ϵ is reduced to 0.03 after 10000 iterations. We have verified that this gives a better steady-state performance and convergence speed for SARSA compared to a fixed value for ϵ. For I-SARSA, no explicit exploration is needed.

Figure 3 shows the evolution of the cost function (i.e., minus averaged reward) over time of SARSA and I-SARSA for the noisy sine traffic pattern. I-SARSA converges to an optimized action-state table in less than 50 iterations, whereas it takes more than 800 iterations for SARSA. Also the performance (after 2000 iterations) of I-SARSA is better than that of SARSA, i.e., 3 [Mbit] versus 50 [Mbit], respectively. The same observation can be made for the fast changing TCP traffic trace. The performance is shown in Figure 4 for a period of 5 seconds (90 s to 95 s) during operation, but it is characteristic for the entire trace. The average performance is 3 times better for I-SARSA than for SARSA.

Figure 5 shows the allocated data rates for SARSA and I-SARSA, for the noisy sine traffic pattern, after only 280 iterations. It shows that I-SARSA succeeds in learning the sine traffic pattern whereas SARSA does not. I-SARSA is very suitable for DRA use cases where traffic patterns can be expected to change quickly.

Finally, Figure 6 shows the Pareto-optimal performance curves for SARSA and I-SARSA, for the noisy sine and TCP traffic patterns. These are the average performances for a varying parameter value β trading-off queuing latency and bandwidth allocation efficiency. The x-axis plots the average queue fill, whereas the y-axis plots the average BW allocation. Ideally you want the smallest possible bandwidth allocation (max. efficiency), while also having the smallest possible queue fill (min. latency), which corresponds to the bottom-left location on the figure. The I-SARSA algorithm clearly outperforms the SARSA algorithm. The Pareto curve of the SARSA algorithm is moreover not consistent. For a changing parameter value β the Pareto curve does not have a convex shape. This is because the learning of the SARSA algorithm is less consistent than that of the I-SARSA algorithm. We have tested different values for parameters α, γ, ϵ but this does not improve the performance of SARSA, whereas we see that I-SARSA always performs very robustly.

Simulation results for other traffic traces (e.g., Speed test, Video call, IoT, File download, ...) show similar benefits for

I-SARSA, but are omitted due to space limitation. I-SARSA has also been compared to commonly considered methods for tackling DRA or traffic prediction problems (ARIMA, exponential averaging, and deep learning solutions). These results indicate that I-SARSA is a very suitable technique for DRA with an improved performance-complexity trade-off, but this content is omitted due to space limitation.

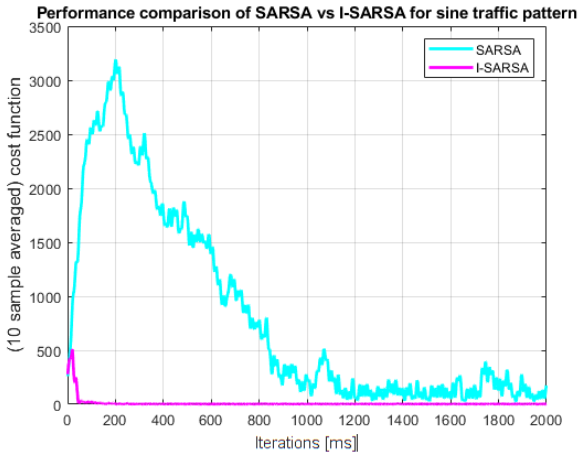

Fig. 3. Avg cost evolution for noisy sine

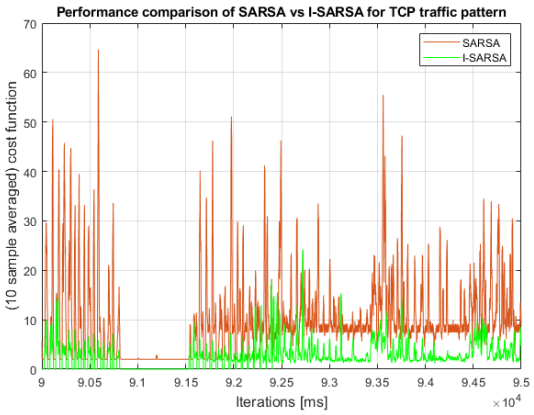

Fig. 4. Avg cost evolution for TCP

VII. Conclusion

We have proposed a novel concept of Inference-based Reinforcement Learning (I-RL) that can be applied to a specific RL problem class that allows to leverage the outcome of a single action to infer the outcome of other actions (that were not performed). This problem class property is exploited to obtain RL algorithms that do not need explicit exploration, which allows to significantly increase the convergence speed, to improve the steady state and transient performance, and to make the operation more consistent and robust. The proposed I-SARSA algorithm has been applied to the problem of dynamic resource allocation, showing large performance improvements compared to existing RL algorithms. We believe that I-RL

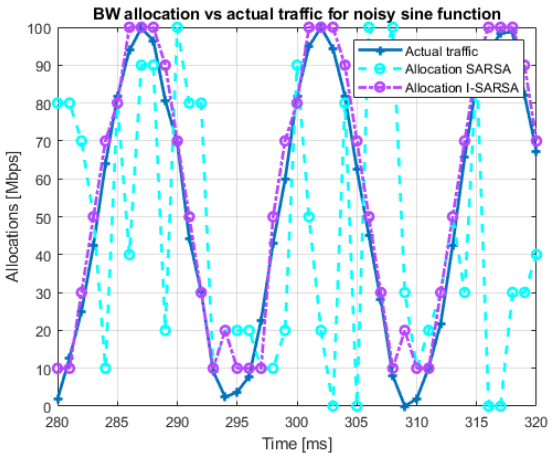

Fig. 5. Learned bandwidth allocations for noisy sine

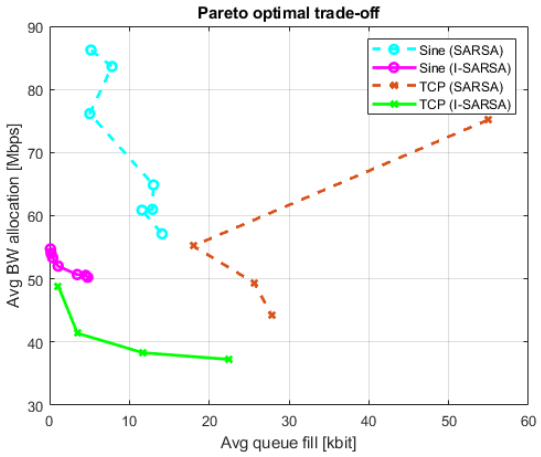

Fig. 6. Pareto trade-offs for noisy sine and TCP traffic patterns

algorithms are especially interesting for very dynamic RL environments, and that the proposed concept is valuable for other signal processing related usecases.

References

[1] N.C. Luong, D.T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, D.I. Kim, 'Applications of deep reinforcement learning in communications and networking: A survey', in IEEE Communications Surveys & Tutorials, Oct. 2018

[2] R. S. Sutton and A. G. Barto, 'Reinforcement learning: An introduction', Second edition, 2018

[3] A.S. Mignon, R.L.A. Rocha, 'An adaptive implementation of ϵ-greedy in reinforcement learning', in International Workshop on Adaptive Technology (WAT), 2017

[4] Y. Shi, Y. E. Sagduyu, T. Erpek, 'Reinforcement learning for dynamic resource optimization in 5G radio access network slicing', in IEEE 25th Int. Workshop on CAMAD 2020, Pisa, Italy

[5] ITU-T Recommendation G.9701, 'Fast access to subscriber terminals (G.fast)', Apr. 2017

[6] J. Maes, C. Nuzman, 'Energy efficient discontinuous operation in vectored G.fast', in Proceedings of IEEE International Communications Conference (ICC), June 2014, Sydney, Australia

[7] ITU-T Recommendation G.9807,'10-Gigabit-capable symmetric passive optical network (XGS-PON)', (2016), Amendment 2 (Oct. 2020)

[8] C. Jin, Z. Allen-Zhu, S. Bubeck, M.I. Jordan, 'Is Q-learning provable efficient', NeurIPS 2018, Montreal, Canada