DISTRIBUTED RESOURCE ALLOCATION IN 5G NETWORKS WITH MULTI-AGENT REINFORCEMENT LEARNING

Jon Menard,
Ala'a Al-Habashna,
Gabriel Wainer

Systems and Computer Engineering
Carleton University
1125 Colonel By Drive, Ottawa, ON, CANADA
{jonmenard,alaaalhabashna}@cmail.carleton.ca
{alaaalhabashna,gwainer}@sce.carleton.ca

Gary Boudreau

Ericsson Canada
349 Terry Fox Drive, Ottawa, ON, CANADA
gary.boudreau@ericsson.com

ABSTRACT

In this paper, we propose using Multi-agent Reinforcement Learning (MARL) for distributed resource allocation in 5G networks. We consider the case where the resource allocation is performed by each User Equipment (UE). The goal will be to learn a joint policy that can be executed by the UEs in a distributed manner. Such policy can achieve a minimum data rate for each user and maximize the sum rate of the users in the network. We consider two different MARL paradigms, namely, Independent Learners (ILs) and Value Function Factorization (VFF). In the latter, we adopt the QTRAN algorithm, which is a value function decomposition-based algorithm that is categorized under the Centralized Training with Distributed Execution (CTDE) regime. Results show that MARL algorithms can be used to learn a joint policy that can be used by UEs for distributed resource allocation.

Keywords: multi-agent reinforcement learning, deep reinforcement learning, 5G, resource allocation.

1 INTRODUCTION

The Fifth Generation (5G) wireless networks support many applications that require high data rates, low latency, and high reliability. For example, ultra-high-definition streaming requires data rates around 25 Mbps with less than 100 milliseconds of latency. Other mission-critical applications such as self-driving cars require 50-100 Mbps but a very low latency of 10 milliseconds. The requirements above have increased the need for radio spectrum.

As the need for spectrum access dramatically increased over the past decade, especially with the introduction of 5G, the radio spectrum has become a valuable and scarce resource. A significant amount of research has been conducted to increase the efficiency of spectrum assignment and utilization. A promising solution to meet the 5G performance requirements is to use multiple tiers in the network architecture with a co-channel deployment scenario (Hossain, et al. 2014) and (Chin, Fan and Haines 2014). In such architecture, in addition to the conventional macrocell-tier with macrocells, there are heterogeneous network tiers that include low-power nodes such as picocells, femtocells, and relays. In addition to the above, wireless peer-to-peer (P2P) communication can be overlaid with the tiers above such as Device-to-Device (D2D) and Machine-to-Machine (M2M) communication, as well as and sensor nodes. Such network architecture would complicate the power and resource allocation problem. There are other scenarios that

add to the complexity of the problem such as Cognitive Radio (CR); a DSA-based approach that allows secondary users to sense the spectrum and exploit vacant spectrum bands without causing significant interference to other users. Other scenarios might involve multiple operators that share the same infrastructure. In such network architectures, the different traffic loads, transmission powers, channel access priorities, and the provision of P2P communication complicate the dynamics of resource allocation.

Recent studies have shown that optimal resource allocation in multi-tier networks is generally an NP-hard problem, and hence computationally expensive. Centralized methods for resource allocation problems in such scenarios are not scalable due to computationally complexity. Furthermore, with the centralized approaches, a centralized node usually performs resource allocation, and such node requires information of the network and its nodes to manage resource allocation, which causes a lot of signaling overhead. On the other hand, distributed or semi-distributed resource allocation methods can provide more efficient solutions for multi-tier networks due to the reduced amount of signaling and computational complexity. In such solutions, multiple nodes can perform resource allocation independently. This includes operator-deployed nodes (e.g., eNBs, relays) or even user devices such as User Equipment (UE) themselves.

Reinforcement Learning (RL) and Deep RL (DRL) have been increasingly used to solve problems in 5G systems such as resource and power allocation. Such algorithms are promising for complicated problems because they allow agents to learn the characteristics of the environment, avoid the exhaustive search in the action space of the problem, and can provide near-optimal solutions to maximize the end-user performance (e.g., SINR and data rate). This is particularly useful in cases with non-convex optimization problems.

However, traditional RL and Deep RL (DRL) algorithms might not be suitable to develop algorithms for distributed execution. Multi-agent Reinforcement Learning (MARL) algorithms can provide a promising solution for such scenarios. MARL includes algorithms for systems that have multiple agents that are interacting within a common environment. Each time step, each agent makes a decision to achieve a predetermined goal that maximizes expected future return. The goal in this case would be for agents to learn a policy such that all agents together achieve the goal of the system. This is very suitable for the case of distributed resource allocation, where UEs need to achieve their data rate requirements and also maximize the performance of the network.

As such, in this paper we propose using MARL for distributed resource allocation in 5G networks. We consider the case where the resource allocation is performed by the UEs themselves.

The goal will be to learn a joint policy that can be executed by the UEs in a distributed manner. Such policy can achieve a minimum data rate for each user and maximize the sum rate of the users in the network. We consider two different MARL paradigms, namely, Independent Learners (ILs) and Value Function Factorization (VFF). In the latter, we adopt the QTRAN algorithm (K. Son, D. Kim and W. J. Kang, et al. 2019b), which is a VFF-based algorithm that is categorized under the Centralized Training with Distributed Execution (CTDE) regime. Results show that MARL algorithms can be used to learn a joint policy that can be used by UEs for distributed resource allocation.

The rest of this paper is organized as follows: Section 2 reviews the background of this work and related work in the literature. Section 3 discusses VFF and QTRAN; the adopted algorithm for resource allocation. Section 4 presents our results, and Section 5 states the conclusion and future work.

2 BACKGROUND AND RELATED WORK

2.1 Reinforcement Learning

Reinforcement Learning (RL) is an area of Machine Learning (ML) focused on sequential decision-making problems in which an agent exists in a mutable environment. The agent can take actions, possibly changing

the state of the environment. When the agent is placed in the environment, it has no prior knowledge, and collects information by interacting with the environment. Each action the agent takes is a decision that is part of the sequential decision-making problem (Sutton and Barto 2018). Deciding which action to take is determined by an action-producing function called the *policy*, π (Graesser and Keng 2019). The *policy* is a function that will output the best action for a particular state of the environment. The reward function determines the value of an action by evaluating the transition between the environment's current state, the agent's chosen action, and the new states (Graesser and Keng 2019). The agent uses the reward to learn good and bad decisions in the environment. As the agent learns, it uses its experiences to adjust its policy for better performance.

Figure 1 shows the agent's interaction with the environment consisting of four main steps to the RL cycle. At each time step, *t*, the agent observes the environment's state ($s_t \epsilon S$). In step 2, the agent chooses an action ($a_t \epsilon A$) and executes it. In step 3, the agent receives a reward, $r(s_t, a_t, s_{t+1}) \epsilon R_a$, evaluated from the current state, current action, and next state, s_{t+1}. Lastly, in step 4, the environment changes its state based on the agent's action, determining the next state, s_{t+1}. The cycle continues, and the agent reinforces its *policy* by learning the states and actions that provide the best rewards.

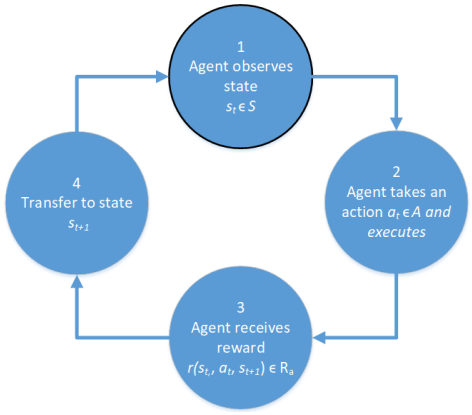


Figure 1: The lifecycle of RL

In RL the agent uses rewards received during training to find the best actions and learn the policy. In a sequential decision-making problem, in addition to the immediate reward, the consecutive rewards from future time steps also matter. The cumulative discounted reward, defined in equation (1), takes into consideration the immediate reward and the reward obtained from next time steps.

$$R = r_{0+}\gamma r_{1+}\gamma^2 r_{2+}\gamma^3 r_{3+} \ldots \gamma^t r_t \tag{1}$$

Gamma, γ, is the discount factor, that takes a value between 0 and 1. It is a crucial variable that controls the importance of future rewards. The larger gamma, the more emphasis on future rewards. As gamma approaches 1, the discounted equation becomes farsighted, and as it approaches 0, nearsighted (Graesser and Keng 2019). With the addition of future rewards, the objective can be represented using the long term cumulative discounted rewards equation,

$$J = \mathrm{E}_{\pi,s_0}[\sum_{t=0}^{\infty} \gamma^t r(\mathrm{s}_t, \mathrm{a}_t, \mathrm{s}_{t+1}) | \mathrm{a}_t = \pi(.|\mathrm{s}_t)] \tag{2}$$

The objective J can be calculated using the average reward over many episodes. The agent's *policy* should maximize the objective, which will require maximizing the reward for each action. However, since the reward function is a part of the environment and unknown to the agent, future rewards can only be determined by predicting how valuable the agent thinks the next state is, not its actual value. The agent uses value functions to quantify the value of the current states and possible actions concerning the expected

future reward. A value function, $V^{\pi}(s)$, evaluates the current states observed by the agent. Using equation (3), the expected return is predicted from the expected trajectory of the current state.

$$V_{\pi}(s) = \mathrm{E}_{\pi, s_o=s}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{3}$$

$E_{\pi,s}$ is the expected value of the return from the state at the current time step, until the state at the last time step. The Q-function, $Q_{\pi}(s, a)$, is the second value function. The Q-function evaluates a state-action pair, i.e., the expected return from being in state s, and taking an action a, as can be seen in equation (4).

$$Q_{\pi}(s, a) = \mathrm{E}_{\pi, s_o=s, a_o=a}[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{4}$$

Similarly, $E_{\pi,s,a}$ is the expected value of the return from being in state s, taking action a, and following policy π until the last time step. An agent's *policy* is a mapped relation between the current observation of the environment and the chosen action described as,

$$\pi: (\mathrm{S} \rightarrow \mathrm{A}) \tag{5}$$

During training, the agent attempts actions and observes the reward obtained. A common approach to facilitate learning and to collect information is through an epsilon-greedy policy to balance exploration and exploitation in the environment (François-Lavet, et al. 2018). Exploration allows the agent to explore more actions of the environment, while exploitation directs the agent towards its perceived maximum reward. The type of action chosen by the epsilon-greedy policy is determined using equation (6).

$$P(Exploration) = 1 - \epsilon \tag{6-a}$$

$$P(Explotation) = \epsilon \tag{6-b}$$

With each action taken, a random variable is sampled and compared to epsilon ϵ. If the random variable is less than epsilon a random action is taken; otherwise, the agent's *policy* determines the following action. Additionally, as training progresses, epsilon slowly decreases until it reaches a pre-defined minimum value. As epsilon decreases, the agent transitions from an exploration-biased action to an exploitation-biased action.

One method for the agent to learn and implement a *policy* is by learning the Q function. The Bellman's equation is commonly used to learn the Q function to find the best Q value for each state-action combination. For example, with Deep Q Network (DQN); a popular DRL algorithm, the Q value for the state-action pair, (s, a), is given by,

$$Q_{\pi}^{*}(s_t, a_t) = \mathrm{E}_{\pi, s'}[r_{s, a, s'} + \gamma \max_{a'} Q_{\pi}^{*}(s', a') | s_t, a_t] \tag{6}$$

According to the equation, the best Q value, i.e., Q^* equals the reward for the current (s, a) state-action pair with the future reward determined by multiplying the gamma value by the maximum expected Q-value of the new states s' when taking action a'. The Q-value mapping for (s', a') is derived from the agent's previous experience.

Different algorithms can be used to learn the Q function. The Q function can then be used to generate a policy. For example, an agent, can use the Q function to choose the action with the maximum Q value. A simple value-based approach to convert the agent's experience into a policy is done through Q learning. At the beginning of training, a tabular representation called a Q table can be initialized (e.g., with zeros) for every cell. With each new state encountered by the agent throughout training, the reward for taking the action is recorded in the Q table. Although this algorithm is simple, it is not guaranteed all state-action combinations will be explored. As such, this algorithm is unsuitable for problems with continuous and high-dimensional action space since the entire state-action space needs to be explored constantly. Furthermore,

unvisited states will have an unknown Q value and cannot outperform random guessing, while visited states will be overrepresented resulting in overfitting (Huang, Chen and Gong 2021).

DQN is an advanced algorithm that involves using a deep neural network (DNN) instead of a table to approximate the Q-value function. The DNN's architecture consists of an input layer to receive the agent's state of the environment, one or more hidden layers, and an output layer to predict the Q values for the actions at that state. The entire DNN uses multiple layers with individual parameters θ, known as weights, to create a mapping of inputs to outputs. When the network receives an input, it propagates the values forward, layer by layer to produce an output. A trained Q network can be used to generate a policy by selecting actions. At each time step, the environment state is fed into the DNN, and the predicted Q values are outputted. The action resulting in the maximum Q value is chosen and executed.

The neural network is trained on the agent's experience by minimizing the error loss between the actual and predicted Q values. While the agent is training in the environment, experience is collected and stored as a tuple (s, a, r, s') into a buffer-replay memory (François-Lavet, et al. 2018). For every N time steps, a sample batch of B experiences from the memory is selected. y_t is computed using Bellman's equation for each sample, and Q_π is calculated using the DNN. Equation (8), shows the network's loss function calculated by finding the mean square error, $E_{s,a}$ from $y(t)$ and $Q(\pi)$. The DNN's parameters are updated to minimize the loss, and training resumes.

$$L_t(\theta_t) = \mathrm{E}_{s,a}[(y_t - Q_\pi(s, s, a, \theta_t))^2] \tag{7}$$

Training the DNN may vary from less than a thousand to multiple million training batches depending on the problem (Graesser and Keng 2019). The DNN can extrapolate the data and develop an optimal policy through training on a subset of the possible environment states. Deep neural networks have state-of-the-art predictive performance, accurately choosing the right action with new unencountered data.

2.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is an adaptation of standard RL where the common environment now includes two or more agents. Agents can have global or individual rewards to promote competitive or cooperative relationships between one another. In competitive environments, each agent learns independently, while in cooperative environments, the agents must work together to satisfy the goal of the system. A global reward is shared between agents whose evaluation is based on the group rather than one single agent. Additionally, an individual reward may exist for individual performance, but not always. Each agent is a learnable unit that aims to develop its policy alongside other agents to maximize the long-term cumulative discounted reward. Due to the non-stationary problem with each agent learning independently and manipulating the environment, training the agents is typically a challenging task categorized as NP-hard problems.

Independent learning is a common method used to implement MARL. All agents are unaware of each other in the environment and learn actions and states individually. One of the most established independent learning algorithms is Independent Q-learning (IQL), proposed by (Watkins. 1989), which uses a Q-function like the previously mentioned ones for its *policy*. IQLs are simple and can be distributed and trained decentralized, allowing for parallel training of multiple agents. Training an independent learner is the same as training a single agent, where the agent performs an action, obtains a reward, and updates their Q-values without considering the actions taken by any other agents. However, the reward is no longer individual and now represents the performance of all agents, not just oneself. With each agent impacting the state of the environment, estimating the Q-value for a state-action pair can be challenging. Independent learning is appealing as it is simple, scalable, and does not require communication between agents. However, IQLs simplicity also exposes some problems. Independent learners' main drawback is their susceptibility to the non-stationary problem. The agent's single point of view depicts a stationary environment, not influenced by other agents. Agents try to optimize their polices while other agents are doing the same. This simultaneous policy update makes the environment intractable and causes the non-stationary problem.

2.3 Related Work

In (Elsayed and Erol-Kantarci 2019), the authors proposed using a Q-learning RL algorithm to balance latency and reliability in Ultra-Reliable Low-Latency communication and throughput for Mobile Broad-Band (MBB) users using 5G networks. Their results demonstrated that the RL algorithm outperformed standard AI fixed power allocation by increasing the throughput 21 times without significantly impacting latency or reliability. Moreover, in (He, et al. 2017), the authors also proposed using a Q-learning RL algorithm to obtain the optimal interference alignment policy in wireless networks. Likewise, the results concluded that the RL algorithm resulted in a significant sum rate and energy performance over previously used methods. In (Shi, Sagduyu and Erpek 2020), the authors used RL for dynamic resource allocation for network slicing in 5G Radio Access Networks. Their results concluded RL outperforms traditional allocation methods well also scaling with an increasing number of UEs. Another proposal by (Mismar, Evans and Alkhateeb 2020) utilizes Q-learning's future reward prediction to improve SINR and sum-rate capacities in sub-6 GHz frequency bands. Similarly, the results favored the RL solution over the industry standards for link adaptation. (Cong and Lang 2021) expand on Q-learning through deep recurrent neural Q-networks (DDRQN) for solving multi-user channel allocation in dynamic multichannel access (DMA). DDRQN can prevent the normal exponential growth associated with each additional channel added, while allowing secondary users to acquire higher spectrum access. As reported in (Cong and Lang 2021), traditional DRL approaches might not be suitable to solve distributed resource allocation problems. MARL algorithms can be used to develop distributed resource allocation algorithms in 5G networks. In this paper, we propose the use of MARL algorithms for distributed resource allocation in 5G networks. We consider two classes of MARL, namely independent learners and CTDE. In the CTDE regime, we adopt the QTRAN framework (K. Son, D. Kim and W. J. Kang, et al. 2019b) built around value factorization function (VFF) to promote joint-action optimization between agents while maintaining decentralized execution without inter-agent communication.

3 CTDE, VFF, AND QTRAN

3.1 CTDE and VFF

An alternative method to independent learning with decentralized training and execution is to use centralized training with decentralized execution (CTDE). With CTDE, each agent has its individualized policy, which uses local observations to compute an individual action (Gronauer & Dieopold, 2022). A decentralized learner's policy is still similar to that of independent learning; however, their optimization is facilitated by the centralized network, not themselves. The centralized network enables information to be universally shared throughout all agents during training, while maintaining partial observability with restricted or non-existent inter-agent communication during execution. The centralized network fully observes the environment, decreasing the non-stationary problem (Gronauer and Dieopold 2022), allowing easier optimization towards a successful joint-action policy. An increasing number of studies have demonstrated CTDE produces successful results in MARL environments (Foerset, et al. 2016) and (Jorge, et al. 2016). One method for training CTDE networks is to use VFF to discretize individual contributions to the joint-action function. VFF allows the agents to learn individual action-value functions, such that optimization for an individual agent leads to optimization of the global joint action-value function. This, in turn, enables agents at execution time to select an optimal action simply by using the individual action-value function, without having to refer to the joint one.

3.2 QTRAN

QTRAN is a recently proposed VFF designed to provide more general factorization than previously proposed methods (K. Son, D. Kim and W. J. Kang, et al. 2019b). For CTDE with VFF, QTRAN interconnects three different deep-neural networks estimators:

- Each agent's individual action-value network $f_q: (\tau_i, u_i) \rightarrow Q_i$

- Centralized joint action-value network $f_r: (\tau, u) \rightarrow Q_{\text{jt}}$
- Centralized state-value network $f_v: \tau \rightarrow V_{\text{jt}}$

Each agent has an individual action-value network used to compute the Q Values for partially observed states. A centralized joint-action value network receives the chosen action from each agent and computes the global Q-values for the joint-action taken. Finally, a state-value network evaluates the state-action combination, ensuring the optimized action-state combination for individual agents matches the optimized action-state combination for the joint action. Training QTRAN starts by gathering experiences for each agent in the environment. After enough experiences have been obtained, a sample batch can be used for training at each time step. Each decentralized agent evaluates its states and predicts the corresponding Q-values. The chosen actions of all agents are passed into the centralized joint-action network which produces its own predicted Q values for the joint action. The loss between the predicted Q values and the actual is combined with the loss between the state-value and action-value networks. The combined loss is backpropagated through the centralized networks and into each decentralized agent's network. Using the centralized networks to calculate loss pushes the decentralized agent's Q values towards the optimal joint action without revealing any additional information to the agents. With each agent's Q value matching the joint action value-network, the agents can be fully disconnected once training is completed and maintain a collaborative joint action even with completely decentralized execution. QTRAN advanced VFF has allowed complex joint optimal actions to be found in MARL environments while previous algorithms have had no success.

4 CASE STUDY

4.1 Simulation Setup

A resource allocation scenario is adopted in this study to evaluate the performance of MARL-based resource allocation in cellular networks. The scenario consists of a single cell that contains N UEs and N available subchannels, with one base-station. The UEs in the cell will be the MARL agents that need to learn a joint policy for resource allocation. Through each episode, agents will be distributed within the range of the cell. Each UE is considered an agent that can transmit on any of the subchannels with two power level: -60 dBm which corresponds to no transmission, and 10 dBm. A UE can transmit on more than one subchannel simultaneously. The agents will use their X and Y coordinates, and their transmission power levels on each of the subchannels as input states to their neural network. While the calculation for path loss, SINR, and spectral efficiency use values in dB (logarithmic domain), the values will be converted to the watts (linear domain) for the agents' observations. The linear domain provides the agent's neural network with a uniform distribution between values, compared to logarithmic, which is harder to estimate. Additionally, each of the states will be normalized between ± 1. Each subchannel will have a bandwidth of 5MHZ. The goal is to maximize the group sum rate, and each agent must achieve a minimum data rate of 40 Mbps at each timestep to solve the environment.

We consider multiple scenarios each with N UEs (i.e., agents), and N subchannels. We use the notation N×N to label a specific scenario. As an example, the scenario for 4 agents and 4 subchannels will be called 4×4.

The agents will use a global reward that is a combination of two factors; the mean data rate which is a factor that is used to maximize the sum rate, and the standard deviation of the data rate which is used to increase the fairness between UEs so that they achieve the minimum data rate requirement. The global reward is calculated by taking the mean throughput of each agent and subtracting the standard deviation as follows,

$$R = \alpha.\mu - \beta.\sigma \tag{8}$$

where μ and σ are the average and standard deviation of the data rates, respectively, and α and β are factors to control fairness among UEs. The only way to maximize this reward is for all UEs to converge towards the maximum mean data rate without any single UE hogging multiple subchannels. Each time a new best

reward is obtained, each agent learns and saves the channel it achieves the best SINR value. The agent's individual reward is calculated based on the data rates achieved on all the available subchannels. If the agent is transmitting on a single subchannel, the reward is the data rate achieved on that subchannel. If the agent is transmitting on more than one channel, the individual reward received is the minimum throughput of any agents in the environment. By receiving the minimum throughput when transmitting to multiple channels, the agent will be directed towards transmitting on the least number of subchannels unless it will not negatively impact any other agents. The individual reward helps guide the agents towards a collaborative solution. The agents will learn not to transmit on multiple channels unless the total throughput increases without negatively affecting other agents.

Each scenario is implemented with our own simulator which was built using Python. The TensorFlow Keras API was used to develop the QTRAN framework (K. Son, D. Kim and J. W. Kang, et al. 2019a). To deploy QTRAN, a central BS would contain the join action-value network and the centralized state-value network, while each UE would have its own action-value network. During training, the UEs will transmit their observations and chosen action to the BS for centralized training. Once the QTRAN network is fully trained, the distributed action-value networks will operate in a decentralized manner and can execute without additional communication with the BS.

4.2 Brute-force Solution

To evaluate the results achieved with the MARL-based algorithms, we compare the results to those achieved with brute-force solution, that performs exhaustive search over all the action space to find the optimal solution for the N UEs (agents) and N subchannels. The action space can be calculated using equation (10).

$$ActionSpace = 2^{N^2} \tag{9}$$

N is the number of UEs (and also the number of subchannels). The action space is a double exponential function and grows substantially faster than a standard exponential or factorial function. The actions space for each attempted configuration is shown in Table 1.

Table 1: Calculated action space for each amount of agents and subchannels

Number of agents	Number of subchannels	Action space
3	3	512
4	4	65,536
5	5	33,554,432
6	6	68,719,476,736

As the brute force algorithm attempts all possible actions, it is significantly impacted by the double exponential growth of the action space. The brute force algorithm solved 3×3 in 0.017 seconds, 4×4 in 2.639 seconds, and 5×5 in 31 minutes and 27 seconds. With the action space expanding rapidly, so does the time taken for the brute force algorithm. The action space for 6×6 is too large, and with our current implementation of the brute force algorithm, it would take 48 days to do the exhaustive search and find the optimal solution. While the brute force algorithm can solve the optimal solution, it is infeasible in this case.

4.3 QTRAN Solution

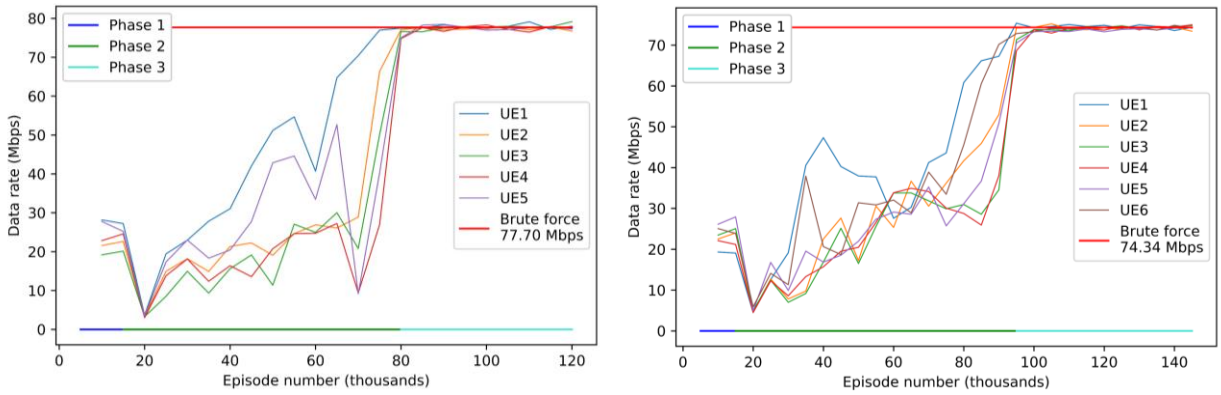
To compare the results achieved with the QTRAN algorithm, we create a baseline with the results achieved using a brute-force solution. The brute-force solution may reach a partial solution by only evaluating a portion of the action space, however this does not guarantee the optimal solution has been found. A partial solution may be used if it is infeasible to search the entire action space, as is in the 6×6 environment where the estimated brute force time to find the optimal solution is 48 days.

A QTRAN model is considered completely trained for the presented scenarios, when the mode performs with over 99% success rate, i.e., the resource allocation results in maximizing the sum rate with all the UEs achieving the minimum data rate in 99% or more of the time steps. At this point, the model is at optimal capacity with an ideal balance between training error and generalization error. Any further training after this point will result in the model overfitting the current environment and preforming worse in any other environment. Given this, at 99% success rate, the models will be considered completely trained, outputting a proper solution to the environment.

Each training figure below shows the training curve of QTRAN for the set scenario. The training cycle can be divided into three distinct phases. In Phase 1, also called the exploration phase, each of the agent's neural network are randomized and untrained. During this phase, agents interact with the environment and take actions to fill the buffer replay without training. In Phase 2, training begins on each of the neural networks. Almost immediately, the throughput for each agent plumets. As the agent's do the first training step on their models, they all converged to taking the same action. With each agent's action being the same, they interfere with each other immensely resulting in almost no throughput. As the training phase progresses, the agents learn to coordinate with each other, and consequently converge to joint policy. Lastly, there is Phase 3. During Phase 3, the agents have reached a plateau in learning, and will not achieve significant performance improvement. Each action results in almost the exact same optimal throughput calculated by the brute force algorithm. At this point the agents can properly solve over 99% of any environment configuration encountered and are considered completely trained.

4.4 Results: Predefined Discrete Number of Locations

With 5 UEs and 5 subchannels, the action space is large. 128 out of 33.5 million joint-action combinations will result in success, allowing for the occasional successful combination to be found during random exploration. Figure 2-a shows the model is able to immediately begin converging after populating the reply buffer and training on the data.



(a) (b)

Figure 2: Data rates of all UEs throughout training (a) 5×5 and (b) 6×6.

The brute force algorithm was able to try all 33.5 million joint-actions in 31 minutes 27 seconds. QTRAN trained over 120,000 episodes and reached over 99% success rate in very close but faster time of 26 minutes. The two models were compared, using the completely trained QTRAN models alongside the brute force algorithm to evaluate 10,000 random configurations of the UEs. The brute force algorithm had a peak throughput of 77.70 Mbps with an average throughput of 77.70Mbps and 0 episodes where the throughput was below 40Mbps for any UE. QTRAN had a peak throughput of 77.70Mbps with an average throughput of 77.679Mbps. QTRAN was able to reach a throughput above 40Mbps for all users in 9,988 out of the 10,000 episodes.

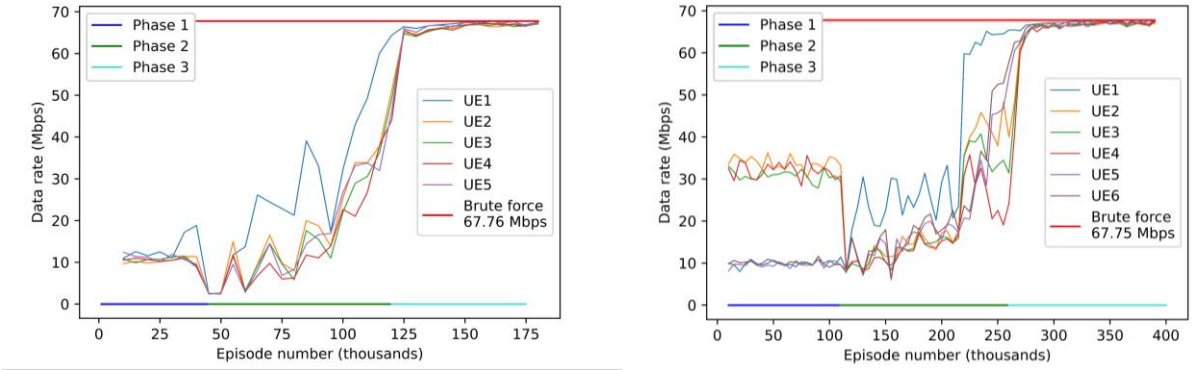
With 6 UEs and 6 subchannels, the action space is significantly larger than the 5×5 with a total size of 68.7 billion joint-actions. Additionally, only 720 joint-action combination would result in all UE's obtaining a throughput above 40Mbps, reducing the odds of finding a successful combination to essentially zero during random exploration. Figure 2-b shows QTRAN is slower than in the case of 5×5 but can still make steady progress towards converging to an optimal policy.

Due to the large action space for this scenario, it was infeasible to search the whole action space with the brute force algorithm to find the optimal solution. After running for 1.1 billion timesteps and exploring only 2% of the possible combinations, the algorithm reached its first successful joint action in 23.65 hours. The time taken to solve the environment with QTRAN is extremely shorter than that of the brute force algorithm. This partial solution is used to benchmark against MARL. QTRAN trained over 145,000 episodes in 38 minutes to reach over 99% success rate and be considered completely trained. The two solutions were compared, i.e., the completely-trained QTRAN model alongside the partial solution found by the brute force algorithm to evaluate 10,000 random configurations of the UEs. The brute force algorithm had a peak throughput of 74.34 Mbps with an average throughput of 74.34Mbps and 0 episodes where the throughput was below 40Mbps for any UE. QTRAN had a peak throughput of 74.34Mbps with an average throughput of 74.31Mbps. QTRAN was able to reach a throughput above 40Mbps for all users in 9989 out of the 10,000 episodes (almost 100% success).

4.5 Results: Randomly-Distributed UEs

In this scenario, the environment will remain the same as previously describe with N agents and N subchannels, with one base station. Unlike the last scenario, the UEs' locations will not be selected based on a list of predefined distances, but rather the UEs will be randomly distributed throughout the cell here at every time step. The base station will still have a range with a radius of 1400m, and all the UE's within the radius but not closer than 50m to the BS. As in the previous scenario, each UE should achieve a minimum data rate of 40Mbps.

As mentioned in the last scenario, with 5 UEs and 5 subchannels, 128 out of 33.5 million joint-action combinations will result in success, allowing for the occasional successful combination to be found during random exploration. Figure 3-a shows the model is still able to immediately begin converging after populating the replay buffer and training on the data.



(a) (b)

Figure 3: Data rates of all UEs throughout training (a) 5×5 and (b) 6×6.

The brute force algorithm was able to try all 33.55 million joint actions in 31 minutes 27 seconds. QTRAN trained over 175,000 episodes with a very similar time in 31 minutes and 33 seconds to reach 99% success rate and be considered completely trained. The completely-trained QTRAN models were compared to the brute-force solution to evaluate 10,000 random episodes with UEs randomly distributed at each episode. The brute force algorithm had a peak throughput of 69.45Mbps with an average throughput of 67.74Mbps and 0 episodes where the throughput was below 40Mbps for any UE. QTRAN had a peak throughput of

69.01Mbps with an average throughput of 67.47Mbps. However, with QTRAN, there were 96 episodes (out of 10,000) where the throughput was below 40Mbps for some UEs.

Figure 3-b shows the training curve for QTRAN in the 6×6 environment. The training cycle is similar to that of the 5×5 scenario, however the training phase was twice as long. With the increased size of the action space for 6×6, the agents were given over double the time to fill their buffer replay. This ensures the models have adequate data to train on, and do not get stuck in a local minimum.

The brute force algorithm's partial solution was found after 23.65 hours, with one single action combination that can solve the environment. It is likely there are more combinations with equal or better performance, but finding a complete solution is infeasible due to the large action space. QTRAN trained over 390,000 episodes in 1 hours 26 minutes to reach 99% success rate and be considered completely trained. The solutions were compared, i.e., the completely-trained QTRAN models alongside the partial brute-force solution to evaluate 10,000 random episode configurations. The brute force algorithm had a peak throughput of 68.80Mbps with an average throughput of 67.75 Mbps and 0 episodes where the throughput was below 40Mbps. QTRAN had a peak throughput of 68.80 Mbps with an average throughput of 67.22 Mbps and 144 episodes (1.44%) where the throughput was below 40Mbps for some UEs.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose MARL-based distributed resource allocation in 5G networks. MARL provides a promising solution to implement resource allocation policies with distributed execution. Using QTRAN, a MARL algorithm that is based on VFF, we demonstrated the advantages of CTDE paradigm for resource allocation, while proving MARL can be used to learn a joint policy executed by UEs for distributed execution. Additionally, we showed a near-optimum solution can be obtained for computationally infeasible problems. We plan to continue researching other CTDE-based algorithms as well as fully distributed algorithms. Moreover, we will explore the use of MARL in other areas of 5G system operation.

ACKNOWLEDGMENTS

This work is funded by Ericsson Canada and the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

Chin, W. H., Z. Fan, and R. Haines. 2014. "Emerging technologies and research challenges for 5G wireless networks." *IEEE Wireless Communications* vol. 21 (2): pp. 106-112.

Cong, Q., and W. Lang. 2021. "Deep Multi-User Reinforcement Learning for Centralized Dynamic Multichannel Access." *6th International Conference on Intelligent Computing and Signal Processing (ICSP).* pp. 824-827.

Elsayed, M., and M. Erol-Kantarci. 2019. "AI-Enabled Radio Resource Allocation in 5G for URLLC and eMBB Users." *IEEE 2nd 5G World Forum (5GWF).* pp. 590-595.

Foerset, J. N., Y. M. Assael, N. Freitas, and S. Whiteson. 2016. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning." *arXiv.*

François-Lavet, V., P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. 2018. *An Introduction to Deep Reinforcement Learning.* Now Foundations and Trends.

Graesser, L., and W.L. Keng. 2019. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python.* Addison-Wesley Professional.

Gronauer, S., and K. Dieopold. 2022. "Multi‐Agent Deep Reinforcement Learning: a Survey." *Artif Intell Rev* vol. 55: pp. 895-943.

He, Y., Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V. C. M. Leung, and Y. Zhang. 2017. "Deep-Reinforcement-Learning-Based Optimization for Cache-Enabled Opportunistic Interference Alignment Wireless Networks." *IEEE Transactions on Vehicular Technology* vol. 66 (11): pp.10433-10445.

Hossain, E., M. Rasti, H. Tabassum, and A. Abdelnasser. 2014. "Evolution Towards 5G Multi-tier Cellular Wireless Networks: An Interference Management Perspective." *IEEE Wireless Communications* vol. 21 (3), pp. 118-127.

Huang, C., G. Chen, and Y. Gong. 2021. "Delay-Constrained Buffer-Aided Relay Selection in the Internet of Things With Decision-Assisted Reinforcement Learning." *IEEE Internet of Things Journal* vol. 8 (12), pp. 10198-10208.

Jorge, E., M. Kågebäck, F. D. Johansson, and E. Gustavsson. 2016. "Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence." *arXiv*.

Mismar, F. B., B. L. Evans, and A. Alkhateeb. 2020. "Deep Reinforcement Learning for 5G Networks: Joint Beamforming, Power Control, and Interference Coordination." *IEEE Transactions on Communications* vol. 68 (3), pp. 1581-1592.

Shi, Y., Y. E. Sagduyu, and T. Erpek. 2020. "Reinforcement Learning for Dynamic Resource Optimization in 5G Radio Access Network Slicing." *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD*).

Son, K., D. Kim, J. W. Kang, D. Hostallero, and Y. Y. 2019a. "QTRAN." May 13. Available via https://github.com/Sonkyunghwan/QTRAN. Accessed March 14, 2022.

Son, K., D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. 2019b. "QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning." *ICML 2019 : 36th International Conference on Machine Learning* vol. 97: pp. 5887-5896.

Sutton, R. S., and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd. London: Bradford Books.

Watkins., C. 1989. "Learning From Delayed Rewards." *Ph.D thesis, King's College, Cambridge*. Available via https://www.cs.rhul.ac.uk/~chrisw/thesis.html. Accessed March 15, 2022

AUTHOR BIOGRAPHIES

JON MENARD is a M.A.Sc student at the Department of Systems and Computer Engineering at Carleton University. He holds a Bachelor's degree in Software Engineering also from Carleton University. His research interests lie in reinforcement learning and their interests regarding wireless communications. His email address is jonmenard@cmail.carleton.ca.

ALA'A AL-HABASHNA is an Adjunct Research Professor at Carleton University. His research interests include 5G wireless communication systems, internet of things, machine learning, computer vision, and modeling and simulation. His email address is alaaalhabashna@sce.carleton.ca and his page is https://carleton.ca/sce/people/al-habashna/

GABRIEL WAINER is Professor in the Department of Systems and Computer Engineering, Carleton University (Ottawa, ON, Canada). His current research interests are related with modeling methodologies and tools, parallel/distributed simulation, and real-time systems. He is a Fellow of SCS. His e-mail is gwainer@sce.carleton.ca. His website is www.sce.carleton.ca/faculty/wainer

Gary Boudreau is 5G Systems Architect at Ericsson Canada. His research interests include digital and wireless communications as well as digital signal processing. His email address is gary.boudreau@ericsson.com