Multi-Agent Reinforcement Learning Methods in 5G Fronthaul Networks

Joshua A. Rutta
Columbia University
Electrical Engineering Department
New York, NY 10027
Email: joshua.rutta@columbia.edu

Joshua D. Libin
Columbia University
Electrical Engineering Department
New York, NY, 10027
Email:jdl2193@columbia.edu

***Abstract*—We propose and tested algorithms that build on previous research that showed average latency in metropolitan computer networks can be reduced by implementing a distributed policy gradient method to assign optical wavelengths dynamically. Specifically, we've investigated two alternate algorithms to demonstrate that latency between Remote Radio Heads (RRHs) and Base Band Processing Units (BBUs) in 5G network architectures can be further reduced. We ran experiments using what we call group actor-critic (GAC) and mini-batch group policy gradient (MB-GPG) reinforcement learning methods to determine resource allocation related to traffic planning and assignment between RRH and BBU pools. These methods yielded an increase in performance under certain environment conditions, although the method proposed previously proved more resilient under higher traffic loads.**

I. Introduction

Conventional cellular networks utilize radio-access networks (RANs) with baseband signal processing done at cellular antenna locations. For 5G networks, separation of the Remote Radio Heads (RRHs) and the Base Band Processing Units (BBUs) has been proposed to improve the scalability of the large numbers of access points resulting from traffic growth and consequent RAN cell sites. In centralized or cloud-RAN (C-RAN), the BBUs are moved to a centralized location, or pool, to allow multiple RHHs and mobile networks to share computing resources, thus reducing monetary expense and energy consumption. The high channel capacity required in C-RAN fronthaul networks motivates the use of wavelength division multiplexed (WDM) optical systems.

Wang, Et al., demonstrated that traffic patterns and loads differ throughout urban landscapes [1], suggesting that allocating traffic among remote BBU pools based on the traffic load could further improve efficiency. This suggested variable allocation is plausibly achieved through utilizing reconfigurable optical add/drop multiplexers (ROADMs) to switch optical paths. Fig. 1 shows a C-RAN architecture with fronthaul WDM connections from an RRH to a remote BBU pool. Although fronthaul enables more efficient cloud based processing, the overall transport capacity requirements increase when the full digitized RF signal is used. Therefore, to enable efficient use of network resources, ROADM optical network reconfiguration and path selection need to be made dynamically.

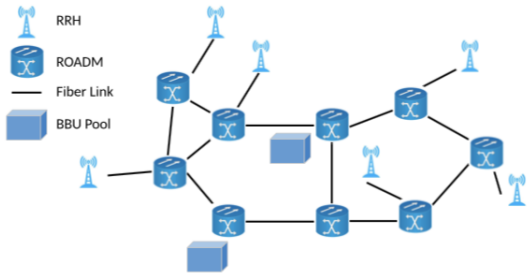

Fig. 1. Central radio-access network (C-RAN) fronthaul architecture with ROADMs used in our simulation.

Dynamic optical wavelength assignment (DOWA) is distinct from conventional data packet routing algorithms inasmuch as the assignments necessitate an end to end circuit from source to destination. Still, DOWA is similar to other resource management problems insofar as resource availability along the fiber and at the BBU pool must be sufficient to initiate the path. In queuing systems job arrival is stochastic and global network resource optimization is often an NP-hard problem. Therefore, the problem of separating RRHs and BBUs in the most efficient manner is modeled as a cooperative multi-agent reinforcement learning algorithm where individaul ROADMs are the agents making decisions on which fiber the wavelength should be be passed to on its way to a BBU pool.

II. Related Work

A. Reinforcement Learning with Computer Networks

Reinforcement learning (RL) methods have been used as early as 1994 in applications to computer networks. So-called Q-Routing [2] used Q-Learning for dynamic packet routing. Recently, the use of RL machine learning methods has exploded. This is partially due to an increase in readily available processing power, and the continually growing amounts of data. Within machine learning, RL has shown been shown to achieve exceptional results in multitudinously disparate applications, such as self-driving cars [3], chess [4], and chemical reactions optimization [5].

The use of software defined networks (SDNs) has enhanced network monitoring and control in wide area networks (WAN) by separating the data and control abstract planes [6]. The control plane informs behavior via a centralized network intelligence which distributes the information through the control plane. This structure will allow each ROADM in our multi-agent reinforcement learning algorithm to view network resources globally. Indeed, a previous course project undertaken by Craig Gutterman showed that using a policy gradient method can reduce average latency in most cases. We've built on his work by employing an actor-critic algorithm and a modified policy gradient algorithm, which yielded more stable convergence to higher rewards and less dropped jobs under certain environment conditions.

III. Problem Formulation

The network we studied is a fronthaul C-RAN network. We represent our network topology in Fig. 1 as a graph whose vertices and edges are, respectively, agents and actions. Vertices represent ROADMs, some of which are connected to BBU processing units or RRHs, axnd edges represent the connecting optical fibers. The channels available are the total number of wavelengths that could be sent simultaneously along a fiber. Our goal is to minimize the average routing time over all jobs in the network by following an optimal policy.

The experiments were conducted on a custom network simulator environment built with OpenAI gym [7] and developed by Mr. Gutterman. Hyperparameters used include the fiber path traversal time, t_{edge}; BBU processing time, t_{BBU}; wavelengths available in each fiber, w; the number of processing units available to each BBU, C_{BBU}; the maximum request lifetime to reach a BBU, μ; and the mean arrival rate, λ, characterizing the Poisson process determining the frequency at which new requests enter the network via an RRH. Each request's destination is unspecified until network arrival, and any BBU pool with sufficient space can be used.

IV. Algorithm Development and Theoretical Results

A. Previous Project Work

In the original project, a design choice was to designate each vertex, which represents a ROADM, as an agent instead of designating a single agent for the whole system. This greatly increases the rate of convergence because the state-action space for a particular agent at each vertex is much smaller and is more rapidly explored than the much larger state-action space of the whole network. Consequently, however, an update for each agent had to be devised that would evaluate a particular agent's actions as well as the actions taken by other agents during that episode, so that a higher reward corresponds to better collective actions taken by the agents. This was captured by setting the reward to the average latency over the jobs in an episode, where the job latency for the k^{th} request is

$$j_k = \begin{cases} (\sum_{edge \in \text{edges visited}} t_{edge}) + t_{BBU} & \text{if path found to BBU} \\ r_{drop} & \text{otherwise} \end{cases}$$

r_{drop} is issued by the environment when an agent assigns a request to an edge with no available channels. We set $r_{drop} = -100$. The reward of a particular episode is

$$r_e = -\frac{1}{n}\sum_{k=1}^{n} j_k$$

which is assigned to every action taken during that episode.

Using this reward, the original project used a modified REINFORCE algorithm to update a policy neural network at each node, where the policy gradient update is

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})(r_e)$$

This update is made over all samples from the agent's history which is shuffled after each episode to reduce correlation. After each episode, the rewards accumulated over the agents history are normalized to reduce variance:

$$\mathbf{r} = \frac{\mathbf{r} - \bar{\mathbf{r}}}{std(\mathbf{r})}$$

We call this algorithm Group Policy Gradient (GPG) to reflect that the input to the neural network on each iteration contains the entire history of all iterations thus far.

B. Current Project Work

We built on these results with two methods. Our first approach was to use actor-critic variant of the previous approach, which we call group actor-critic (GAC). This was achieved by introducing a second neural network for every policy network to play the role of a critic network, which outputs a value given a state. This critic was used as baseline for the gradient update to the policy network. The update to the critic network was the gradient of an L_2 loss with the output regressed against the reward defined previously:

$$\mathcal{L}(\phi) = \frac{1}{2}||\hat{V}_\phi^\pi(s) - r_e||^2$$

The update to the actor network was as follows:

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})(r_e - \hat{V}_\phi^\pi(s))$$

Since $\hat{V}_\phi^\pi(s)$ is only a function of s, this yields an unbiased baseline to reduce the variance. An additional change we made to this model was that our gradient update only used actions sampled from the past episode. In addition to improving the policy, this also had the added benefit of reducing the run time of each experiment by more than than a factor of 10 and decreasing the memory required to run the agents.

After seeing the dramatic improvement in results, we implemented this strategy of only using the data from the past episode to the policy gradient approach, which we call mini-batch group policy gradient (MB-GPG), and found that this slightly outperformed GAC, converging more quickly to optimal behavior.

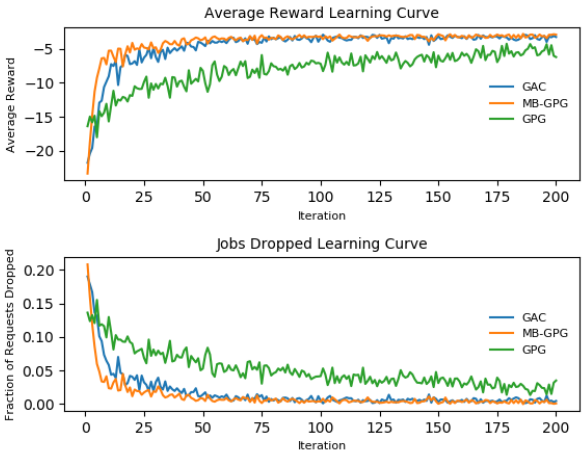

Fig. 2. Comparison of average reward and dropped jobs learning curves for three algorithms studied at arrival rates of 5 milliseconds. The legend entries are GAC, GPG, and MB-GPG corresponding to group actor-critic, group policy gradient, and mini-batch group policy gradient. At 5 ms the MB-GPG, and GAC outperformed GPG in convergence to least latency.

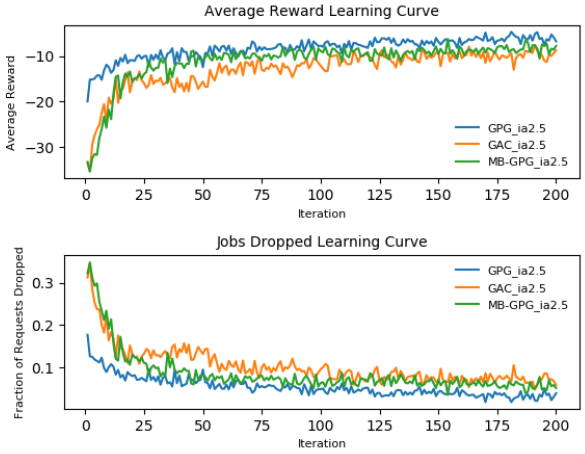

Fig. 3. Comparison of average reward and dropped jobs learning curves across three algorithms studied at arrival rates of 2.5 milliseconds. The legend entries are GAC, GPG, and MB-GPG corresponding to group actor-critic, group policy gradient, and mini-batch group policy gradient. At 2.5 millisecond arrival rates GPG outperformed both our algorithms.

Fig. 4. Comparison of average reward and dropped jobs learning curves across three algorithms studied at arrival rates of 1.25 milliseconds. The legend entries are GAC, GPG, and MB-GPG corresponding to group actor-critic, group policy gradient, and mini-batch group policy gradient. At 1.25 ms the GPG and GAC algorithms converge more slowly than GPG, and to higher latencies.

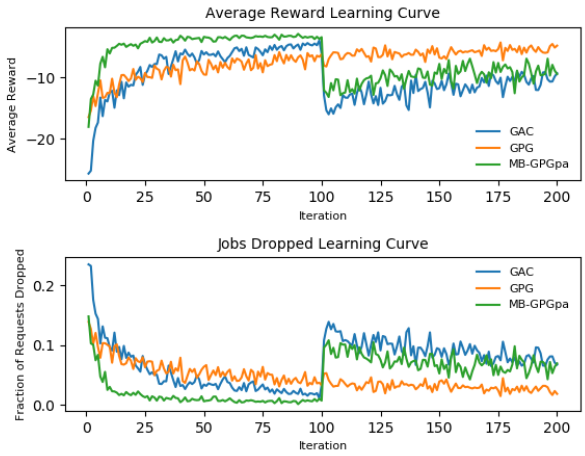

Fig. 5. Comparison of average reward and dropped jobs learning curves for three algorithms studied at epochs where arrival rates start at 5 milliseconds and change half way to 2.5 milliseconds. The legend entries are GAC, GPG, and MB-GPG corresponding to group actor-critic, group policy gradient, and mini-batch group policy gradient. Note, that the GPG does not change drastically at the transition point

V. Experiment Parameter Settings

The environment hyper-parameters used are presented in Table I. With these hyper-parameters we executed each algorithm three times changing only the mean inter-arrival time of requests.

We changed the Poisson process to have mean inter-arrival times of 5, 2.5, and 1 milliseconds. Results are presented graphically in figures 2, 3, and 4.

We used two neural networks; one to output a policy distribution for each node and the other to output a scalar value function to evaluate the states of the network. The policy neural network (PNN) was used for policy updates which occurred in both the actor-critic and modified policy gradient methods. The actor-critic, however, also utilized the value neural network (VNN). The neural network architectures are now discussed:

TABLE I
ENVIRONMENT HYPER-PARAMETER SETTINGS

Hyper-Parameter Symbol	Setting
$\mathbf{t_{edge}}$	0.5 ms
$\mathbf{t_{BBU}}$	2.7 ms
$\mathbf{w}$	10 channels
$\mathbf{C_{BBU}}$	30 processing resources
μ	10 ms

cc

A. Neural Network Architectures

We use three distinct neural networks in the course of our work. Two of these neural networks output policies and the remaining neural network outputs a single scalar value that is used as the critic for the state.

B. PNN for MB-GPG

Mini-batches of size 50 are passed into a fully forward connected neural network with depth three (3 hidden layers). Details are now presented

- Inputs - size 50 mini-batch of state action observations
- Depth - 3
  - First Hidden Layer
    - 25 ReLU activation units
    - weights initialized according to a normal distribution
    - biases initialized to 1
  - Second Hidden Layer
    - 15 sigmoid activation units
    - weights initialized according to a normal distribution
    - biases initialized to 1
  - Third Hidden Layer
    - n ReLU activation units, where n is the number of actions possible for a particular agent. n is 2 or 3 for any given agent in our network.
    - weights initialized according to a normal distribution
    - biases initialized to 1
- Outputs - policy distribution for that node given the state
- Update Technique Used - Adam Optimizer; more computationally efficient version of general stochastic gradient descent.

C. PNN for GPG

This PNN has the same architetecutre as the PNN for MB-GPG, with one difference major difference. The inputs are entire batches of each iteration thus far. For example, if iteration one has x data points to fit, and iteration two also has x points to fit, then iteration two fits 2x data points, and iteration three fits 3x data points, and so on.

D. PNN for GAC

The architecture is identical to the architecture of the PNN for MB-GPG.

E. Value Neural Network for Actor-Critic

The architecture is identical to the architecture of the PNN for GAC *except* the second layer uses ReLU activation functions, and the last layer uses linear activation functions. Also, the output is a single scalar value for the entire network, for all nodes.

VI. EXPERIMENT RESULTS

From these figures it can be seen that for a mean inter-arrival time of 5 ms, which was the setting under which Mr. Gutterman presented the results in his report, our MB-GPG algorithm converged more rapidly and with higher average reward than the GPG method employed in previous work. The same can be said for GAC, although that was outperformed by MB-GPG. However, at higher loads, such as those with mean inter-arrival times of 1.25 ms, GPG outperformed both GAC and MB-GPG with a much larger differential than either of our modifications outperformed GPG with low loads. We hypothesize that this is because in a network with higher loads and much more volatility, a history of this volatility is more important to inform an optimal policy than when the network is relatively calm. More testing would have to be conducted to see which method would be of more utility in real life applications, because GPG took approximately four hours to execute on our toy model, which implies it may take a prohibitively long time to run on a real network.

Although we didn't have time to test it, we propose further investigation to be done on trying to find a medium between GPG and MB-GPG, perhaps with a dynamic batch size that increases as traffic load increases, to reap the benefits of robustness under higher traffic loads of GPG and the quick training time and low memory requirements of MB-GPG.

Fig. 5, shows how the three algorithms respond to non-stationary processes, where we changed the mean inter-arrival time midway through the simulation. At 1000 iterations the mean inter-arrival time changes from 5 milliseconds to 2.5 milliseconds. While our algorithm's rewards sharply change at 1000 iterations, GPG's rewards continues its trajectory without drastic changes. We hypothesize that this is due to the fact that GPG is trained with a much larger data set, so the policy is resilient to sharp changes introduced by incoming data. We were motivated by the differentials apparent in Fig 3 and Fig 4 to examine the algorithm's performances' in a network environment with non-stationary arrival rates. In Fig 3 GPG, GAC, and MB-GPG are very similar with GPG performing best. In Fig 4, GPG, GAC, and MB-GPG are also similar in shape, *but* the differential from GPG to GAC and MB-GPG is comparatively larger than the differentials that can be seen in Fig 3. We note that at very low arrival times, like 10 ms, all algorithms converged rapidly to the optimal policy, well before 1000 iterations are met. We did not empirically observe how the algorithms would respond to sudden arrival

time changes, but we expect that there would be no drastic change in average reward and dropped jobs based on our knowledge that our algorithms can handle low loads and that GPG performs exceptionally at high loads.

VII. Conclusion

Our algorithms converged more rapidly than GPG, and are perhaps more realistic for real life applications because of their execution time, despite GPGs significant advantage in performance optimality in high load networks. As previously mentioned, we propose further investigation to be done on trying to find a medium between GPG and MB-GPG, perhaps with a dynamic batch size that increases as traffic load increases, to reap the benefits of robustness of GPG under varying traffic loads and generally higher traffic loads, and the quick training time and low memory requirements of MB-GPG and GAC.

Acknowledgments

We would like to thank Craig Gutterman, for his guidance and inspiration on this project idea. Mr. Libin would like to thank his wife for her unwavering support through his long hours of work.

Team Member Contributions

Both team members made significant contributions to this project. Mr. Rutta implemented the GAC and MB-GPG methods with assistance from Mr. Libin, who was also essential in organizing the code's readability, as well as using git version control software to seamlessly allow the team to work collaboratively and remotely. Mr. Libin produced the graphs used in this report, and both contributed to writing and editing the body of this report.

References

[1] H. Wang, F. Xu, Y. Li, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: ACM, 2015, pp. 225–238. [Online]. Available: http://doi.acm.org/10.1145/2815675.2815680

[2] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994, pp. 671–678.

[3] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J. Allen, V. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," *CoRR*, vol. abs/1807.00412, 2018. [Online]. Available: http://arxiv.org/abs/1807.00412

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[5] Z. Zhou, X. Li, and R. N. Zare, "Optimizing chemical reactions with deep reinforcement learning," *ACS Central Science*, vol. 3, no. 12, pp. 1337–1344, Dec 2017. [Online]. Available: https://doi.org/10.1021/acscentsci.7b00492

[6] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 87–100. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934904

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540