Multi-Armed Bandits for Optimizing New Peers in Peer-to-Peer Networks

Oscar Sandford [1] **Shawn Nettleton** [1]

Abstract

Pure peer-to-peer networks serve to secure information in a decentralized, distributed topology. The multi-armed bandit (MAB) problem formulation proves to be a useful tool for abstracting the problem of optimizing new peer connections. Based on their use in related works, we survey and evaluate several MAB algorithms (including variants of ε-greedy, UCB, and SoftMax) in the context of identifying the fastest peer to download from during the connection process. UCB and ε-first performed the best at selecting the optimal peer in each of our test scenarios. In a condensed period, SoftMax underperforms in comparison, but eventually overtakes ε-greedy.

1. Introduction

Peer-to-peer computer networks create a unique environment for content distribution wherein the integrity of the system is not compromised by the failure of a single, centralized node in the network. According to (Schollmeier, 2001), true peer-to-peer systems require peers to be mutually directly accessible (without intermediate entities), as well as the network state or quality of service being preserved in the advent of a peer being removed from the network, for any reason.

The requirements for peer-to-peer networks in different application domains vary. However, new peers that are directly accessing the server for the first time have no information on the network state. New peers therefore cannot be held accountable to preserve the network state and its content if other nodes disconnect. It is essential that this new peer is fed the relevant data as fast as possible in order to fulfill both the requirements of a true peer-to-peer environment, as well as any necessary quality of service targets. With the added volatility of a dynamic network setting, the rate at which a new peer can be brought "up to speed" becomes far more crucial.

In this study, we abstract the new peer scenario described above as a reinforcement learning problem with multi-armed bandits. The multi-armed bandit problem involves k slot machines (slot machines are sometimes called one-armed bandits) which pay out reward values according to an internal distribution, of which the agent is not aware (note that some MAB variations may relax these constraints). The goal is to pick a strategy to learn which arms pay out the most in order to to maximize total reward over a set number of rounds (Vermorel & Mohri, 2005).

We consider various algorithms to solve the multi-armed bandit problem, and a select few are implemented in order to evaluate their efficacy against this problem. Related literature is surveyed in order to compare our work with solutions to similar problems and verify the validity of our results. Formulation of this challenge as a reinforcement learning problem precedes an explanation of the approach and a discussion of the results. The following section contains a survey of related work concerning the application of multi-armed bandits to computer networking problems.

2. Related Work

Multi-armed bandits serve as a useful abstraction for optimization problems that require decision making with reward outcomes that are initially unknown. In a study on cognitive radio networks (Modi et al., 2017), secondary user nodes select a single channel for information exchange at one time, with no knowledge about channel quality or availability. The authors use a variation on the upper confidence bound (UCB) algorithm, namely QoS-UCB. Their scenario is called "restless", meaning that the states of the arms can fluctuate over time, affecting their internal distrubtions and the resulting payouts.

The task of wireless network selection, with the goal of maximizing perceived quality for the end user, is handled by extending the bandit model to be more flexible (Boldrini et al., 2018). In this formulation, the agent can take one of two actions (which can span multiple time steps): measure or use. The difference is that measurement allows only evaluation, whereas using adds exploitation. Measuring takes less time than using, which can span a set number

[1]Department of Computer Science, University of Victoria, Victoria, Canada. Correspondence to: Oscar Sandford <oscarsandford@uvic.ca>, Shawn Nettleton <shawnnettleton@uvic.ca>.

RL Course Project Report, University of Victoria, Spring, 2022. Copyright 2022 by the author(s).

of time steps. Results showed that the choice of algorithm depended on the payout distributions. Conservative UCB1 is useful when arm rewards are similar, MLI when one arm is clearly better. More aggressive algorithms like POKER can lead to low regret but high variability, and are therefore less reliable (Boldrini et al., 2018).

Anver and Mannor share methods for multiple multi-armed bandit agents, coordinating with each other and learning stochastic network conditions, which may vary between users (Avner & Mannor, 2019). Their problem formulation is similar to our intentions, but with the agent transmitting instead of receiving. Further, they bound their rewards on the interval $[0, 1]$. The problem with this reward formulation when it comes to receiving is that, while outward transmission speed or success may be measureably bounded, reception rate is not necessarily bounded. In fact, there may be conditions when the end receiver does not have the resources to unpack the transmission packages in time, and will become congested. This paper uses techniques to deal with collisions when two or more users transmit in a single channel (Avner & Mannor, 2019). In our problem, we are only operating with unicast in a channel selected by the requesting peer. A last thing of note is Anver and Mannor's use of UCB in the channel ranking part of their algorithm selection.

Another paper surveys resource scheduling with multi-armed bandits in wireless networks (Li et al., 2020). They mention that ε-greedy, an algorithm that balancing exploration and exploitation, has shortcomings in its "pure" randomness, and does not take into account confidence intervals on the reward estimates of each arm. UCB exploits this, and also tapers off exploration over time. The authors make the distinction between single- and multi-player multi-armed bandits (SMAB and MMAB), where the former involves a single agent operating the bandit selection mechanism. SMAB have applications in our single peer leeching scenario, as well as centralized network algorithms. MMAB often involve distributed selection that sacrifices independence for synchronization overhead (Li et al., 2020).

Hillel *et al* explore how k players collaborate to identify an ε-optimal arm in a MAB setting, and determine communicating only once players are able to learn at a rate of $\sqrt{k}$ times faster than a single player (Hillel et al., 2013). This methodology may prove useful as the number of peers within the network fluctuates or quality of service changes.

Multi-armed bandit problems have been gaining significant attraction. However, little effort has been devoted to adapting existing bandit algorithms to specific architectures such as peer-to-peer network environments (Szörényi et al., 2013). This paper successfully implements the ε-greedy stochastic algorithm to a peer-to-peer network environment, scaling with the size of network, achieving a linear speedup in terms of the number of peers, and preserving the asymptotic behaviour of the standalone version. They also present a heuristic which has a lower network communication cost which may prove helpful with adapting other related works.

Scenarios of distributed clustering have shown to be promising at solving MAB problems within peer-to-peer networks (Korda et al., 2016). There are two setups in particular, the first where all peers are solving the same problem and second where a cluster of peers solve the same problem within each cluster. This has shown to achieve an optimal regret rate, defined as the expected difference between the reward sum associated with an optimal strategy and the sum of the collected rewards.

Competition amongst peers is inevitable within the peer-to-peer network environment, especially when peers are trying to stay up to date. Managing this competition can be difficult given its unpredictable nature. Miao Yang *et al*, develop an online learning policy based on top of a MAB framework to deal with peer competition and delayed feedback (Yang et al., 2020). However, their work is with relation to fog networks, a decentralized computing infrastructure between the data source and the cloud. We aim to utilize some of their understandings while tackling the peer-to-peer network scenario.

The work done within (Si et al., 2008) showcases new considerations for not only optimizing data rate transmissions in wireless peer-to-peer networks but also minimizing power consumption. Similar limitations are becoming popular when analyzing the use of graphical processing units (GPUs) such as the work within (Tasoulas & Anagnostopoulos, 2019). As more domain problems are aspiring for optimal performance it is important to recognize new aspects such as power consumption.

3. Problem Formulation

Although we have so far informally hinted at the application of multi-armed bandits to this problem, this section is devoted to formalizing that congruence.

Consider the setting of a peer-to-peer network wherein a new peer joins with the intent to be brought "up to speed" with the rest of the network as soon as possible (i.e. download all the data in the network from other peers). However, the new peer does not know the network speeds of its seeds, just how much data it receives over time when it chooses a peer and receives data from them for one (or more) time step(s). The reward is the average number of bytes received across the number of time steps spent on that action. We will assume that data packets are UDP datagrams.

Do not confuse time steps with steps (or rounds) in the algorithm. Multiple time steps can happen during a single round, because *a step is when the agent makes a decision for a given number of time steps*. We use steps and rounds interchangeably. Keep this distinction in mind in the context of this study.

The agent should aim to choose the peer that is transmitting the fastest. However, consider that network speeds may change, and the optimal seed to leech from will not always be the best. We call this a "restless" scenario. We simulate these dynamics by assigning each peer a set of possible states, as well as a transition matrix of probabilities to transition from one state to another at every time step. In this sense, every peer is running its own Markov process in the background, irrespective of the agent's action.

This scenario can be solved with a multi-armed bandit approach. Each peer is considered as an "arm" in the multi-armed bandit algorithm. The agent will choose to "pull an arm" and receive reward for a certain number of *time steps* (by default 1). During every time step, the network dynamics shift, and each arm may transition to another one of its states (regardless of if it was the arm selected or not). Naturally, this creates greater variance in average reward payout, which serves to simulate the noise present in real-world network systems.

Previous studies (Avner & Mannor, 2019; Szörényi et al., 2013; Yang et al., 2020) consider peer-to-peer networks where the peers communicate or compete with one another. However, our scenario assumes no information about peers is initially present on the hardware of the new peer, and that transmitting this information ahead of the vital data packets should not be a priority. Therefore, the "trial-and-error" methodology of multi-armed bandit agents fits the learning requirements under these constrained conditions.

4. Approach

In this section we briefly outline each algorithm considered, and present our approach for generating test scenarios and measuring the algorithms' efficacy against the new peer update task within a small set of hyperparameters. The accompanying source code for our work is maintained on GitHub[1].

4.1. Algorithms

The most common baseline algorithm applied to multi-armed bandit problems is called ε-greedy. ε-greedy takes a single hyperparameter ε that dictates the probability of exploration (i.e. choosing a random arm from the possible selections), with $1-\varepsilon$ being the probability of choosing the optimal arm based on the average rewards so far. We make use of ε-greedy, as well as some of its variants in this study.

[1]https://github.com/oscarsandford/network-bandit

Firstly, ε-first is an ε-greedy strategy where only exploration is done for the first $T\varepsilon$ rounds, and pure exploitation is done during the remaining rounds. T is the number of rounds (also called steps) per run (Vermorel & Mohri, 2005). This forced exploration means peak rewards will be delayed, but broader experience is gained as a tradeoff.

In addition, ε-decreasing is another ε-greedy variant wherein the initial ε value ε_0 is decreased over the number of rounds completed. More specifically, the probability of exploration at time t is given by $\varepsilon_t = min\{1, \frac{\varepsilon_0}{t}\}$ where $\varepsilon > 0$. Values for ε_0 are typically not on the interval $[0, 1]$, instead values like 1.0, 5.0, and 10.0 are used (Vermorel & Mohri, 2005).

SoftMax (also called Boltzmann Exploration) makes action decisions based on *probability matching* methods (Vermorel & Mohri, 2005). Each arm a of k arms holds an associated probability $p_a = e^{Q_a/\tau} / \sum_{a'} e^{Q_{a'}/\tau}$ where Q_a is the estimated action value associated with action a. The hyperparameter of SoftMax is τ, called the *temperature*. It can be varied similar to ε in ε-greedy.

Exp3 is a variant of SoftMax, using the idea of Boltzmann Exploration and probability matching (Vermorel & Mohri, 2005). Each arm a of k arms has an associated probability $p_a(t) = (1-\gamma)\frac{w_a(t)}{\sum_{j=1}^{k} w_j(t)} + \frac{\gamma}{k}$ of being pulled at time t. The single hyperparameter γ indicates the learning rate. The weights associated with each action a at time t are denoted $w_a(t)$.

UCB (Upper Confidence Bound) and slight alterations of UCB were used in many of the previously mentioned works. Given its popularity and excellent results we hoped it would also perform well within our environment setting. Instead of selecting actions based on probabilities, UCB alters its exploration and exploitation as it gathers information about the environment. Least taken actions will be prioritized during the exploration phase, and once the estimated action values are more established, UCB will exploit the action with the highest estimated reward. This action selection is derived from the following: $A_t = argmax_a \left[Q_t(a) + C\sqrt{\log t / N_t(a)}\right]$ where $Q_t(a)$ is the estimated value of action a at time t, C is a confidence value hyperparamter that controls the level of exploration, and $N_t(a)$ is the number of times action a has been selected prior to time t. $Q_t(a)$ is the exploitation part of the equation, while the second half handles the level of exploration also controlled by the hyperparameter C. UCB-1 slightly alters the default algorithm by including a constant of 2 being multiplied by the $\log t$ as such, $A_t = argmax_a \left[Q_t(a) + C\sqrt{2\log t / N_t(a)}\right]$. This is a popular take on the original UCB algorithm that we thought to include to see how they may differ in our implementation.

Algorithm 1 create_peers

```
Input: number of peers $n$, the distribution for state counts $\mathbb{S}$ and its parameters $s_p$, the distribution for means $\mathbb{M}$ and its parameters $m_p$, and the distribution for standard deviations $\mathbb{D}$ and its parameters $d_p$
Output: a list of $PeerArm$ objects $\Phi$
Initialize $\Phi$ as an empty list
for _ = 0 to $n$ do
    $n_s \sim \mathbb{S}(s_p)$
    if $n_s < 1$ then
        $n_s = 1$
    end if
    Initialize $\Pi$ as an empty list of means
    Initialize $\Sigma$ as an empty list of standard deviations
    for _ = 0 to $n_s$ do
        $\pi \sim \mathbb{M}(m_p)$
        $\sigma \sim \mathbb{D}(d_p)$
        Append $\pi$ to $\Pi$
        Append $\sigma$ to $\Sigma$
    end for
    Initialize $T$ as an empty $n_s \times n_s$ transition matrix
    for i = 0 to $n_s$ do
        Sample $n_s$ values $\rho_i \sim Uniform(0, 1)$
        Set each $T_{i,j}$ to $\frac{\rho_{i,j}}{\sum_j \rho_i}$ $\forall j \in [0, n_s)$
    end for
    $\phi = PeerArm(\Pi, \Sigma, T)$
    Append $\phi$ to $\Phi$
end for
Return $\Phi$
```

POKER was shown by (Boldrini et al., 2018) to have low regret but be unreliable due to high variance, due to these problems, time constraints, and the complexity of the pseudocode included in the paper, we deigned to not cover POKER in our evaluation.

4.2. Implementation

Our baseline peer setup for this assessment is simply a set of 5 single-state peers we call $base5$: $PeerArm(2, 1)$, $PeerArm(4, 1)$, $PeerArm(6, 1)$, $PeerArm(8, 1)$, $PeerArm(10, 1)$. This simple layout removes the simulated dynamism of multi-state peers, but provides a good baseline evaluation for each algorithm.

In order to automate the creation of multiple peers, we devised Algorithm 1 which takes as input a number of peers to generate, and three distributions (with their associated hyperparameters) from which to sample the number of states a peer will have, and each state's mean and standard deviation reward. Each state requires a mean and standard deviation because rewards are generated using a normal distribution.

The $gen10$ set of peers uses Algorithm 1 to create 10 peers using the specification below, each of which is more realistic in their transmission speed (reward) variance than $base5$. We also do similar for 20 peers, naturally labelled $gen20$.

```
gen10 = create_peers(10,
   np.random.poisson, dict(lam=5.0),
   np.random.normal, dict(loc=10.0,
       scale=1.5),
   np.random.normal, dict(loc=0.5,
       scale=0.1))
```

The algorithms discussed in section 4.1 are implemented in Python and analyzed using a Jupyter Notebook. Each algorithm makes use of a generic $BanditEnv$ environment in order to execute actions and reset the environment after each run.

The action value estimates are computed by using a sample-average estimate of action value, with an initial estimate of 0 for each peer. This method does not take advantage of the stochastic state-changing Markov process in each peer, as the agent would have to learn the transition matrix for each peer as well. In the interests of time and leaving some more on the table for further research, we stick to the sample-average estimate method for this study.

By default the MAB algorithms will take an action for 1 time step, but we have added functionality such that it is possible to take more than 1 action, and the end reward for that step is simply the average of rewards received by taking that action for that many time steps. Recall that peers' state transitions will continue to be active for each time step. That is, the agent cannot "lock" the network state by taking an action for several time steps.

5. Results

Each algorithm completes 100 runs, with a fixed number of steps each. The number of steps relates to how long the peer will be receiving data from its peers. This number is varied in order to visualize the algorithmic performance in the short- and long-term.

The average reward for each step is averaged across 100 runs, and each of these averages is plotted across the number of steps. In the following two subsections, we briefly discuss our prior assumptions on the results, and then present plots containing visually definitive evaluations of each algorithm in our test environment.

Our experiment results do not include UCB-1. Due to the similarities to UCB, UCB-1 produces almost identical results. This was not unexpected, and it was excluded in order to simplify the comparisons with the other algorithms . We also decided to drop ε-decreasing, as its poor performance produced nothing of note.

5.1. Theoretical Assumptions

The preceding literature made ε-greedy out to be an attractive baseline that fares well for many use cases. We had little expectation for the ε-greedy variants, ε-first and ε-decreasing, and expected them to perform somewhat similar to each other. Algorithms based around UCB were commonly used in related works, and we therefore expected UCB to be stronger than ε-greedy. SoftMax was expected to be eponymously "softer" than ε-greedy, and approach its maximum slower but have less variance. Exp3 was expected to be similar to SoftMax due to it being derivative of the SoftMax probability drawing method.

5.2. Experimental Results

Figure 1 shows the general performance of each algorithm when it comes to the set of static peers, $base5$.

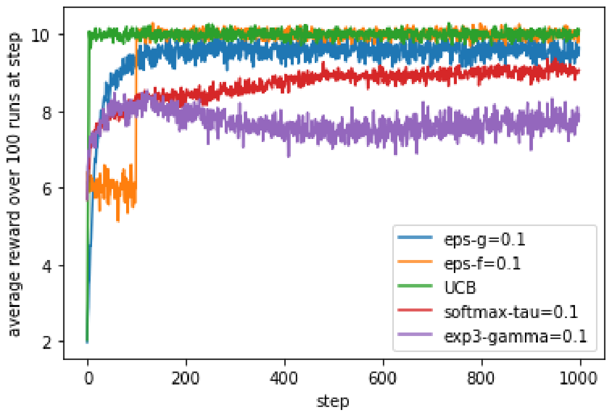


Figure 1. Average reward over time for each algorithm over 1000 steps using the $base5$ peers with 1 time step per action.

The papers we referenced seem to agree that ε, τ, and γ values should be around 0.1. However, we decided to see if stronger exploitation would benefit the learning curve. We noticed that lowering UCB's C hyperparameter did not improve its already strong performance at $C = 1$, so we increased it instead. Figure 2 uses the $base5$ peers with UCB's C value set to 15, and all other algorithm hyperparameters set to 0.01.

Figure 3 uses the dynamic $gen10$ peer set. Exp3 seems to peak higher than SoftMax compared to Figure 1, but then decays below SoftMax's performance. The other algorithms have similar performance, although it seems that UCB's dominance suffers only slightly in stochastic scenarios.

Cranking the time steps to 20 (the amount of time steps the agent will dedicate to a particular peer) shows a slight reduction in the variance of SoftMax and Exp3 in Figure

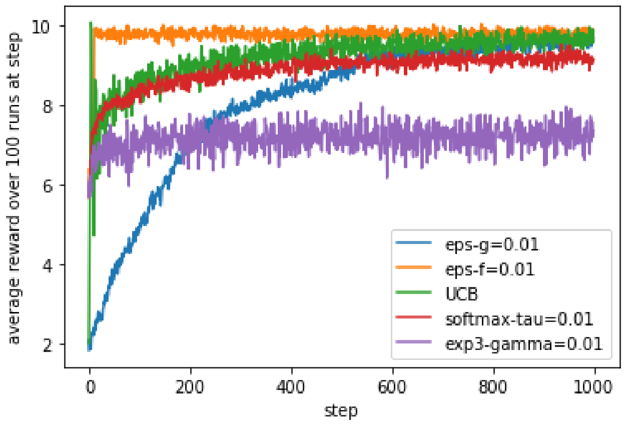


Figure 2. Average reward over time for each algorithm over 1000 steps using the $base5$ peers with 1 time step per action. Different hyperparameters are noted. UCB uses $C = 15$ as opposed to $C = 1$.

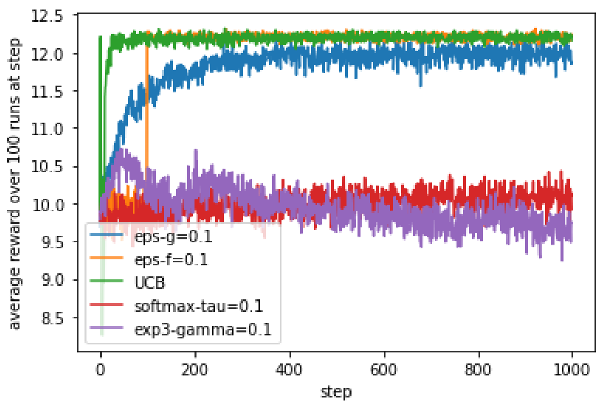


Figure 3. Average reward over time for each algorithm over 1000 steps using $gen10$ peers with 1 time step per action.

4. However, it is also worth noting that the overall average reward peaks lower for each algorithm, compared to Figure 3, with 1 time step per action.

Based on the results in Figure 1, we wanted to find out if SoftMax would ever overtake ε-greedy. In Figure 5 we see that SoftMax eventually stabilizes higher than ε-greedy, and with lower variance.

However, Figure 6 shows that this is not the case for the more dynamic $gen10$ peers. Even after only 20000 steps, it is clear that SoftMax is not rising above ε-greedy like it was with $base5$.

As we have seen, ε-first is capable of competing with UCB in optimal peer selection. In Figure 7, we show that extending the exploration proportion for more peers does not

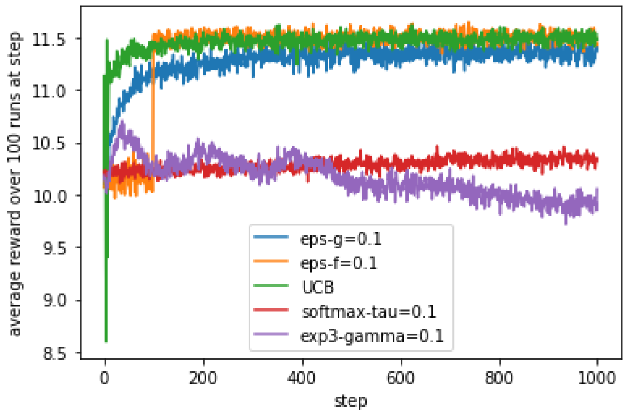


Figure 4. Average reward over time for each algorithm over 1000 steps using $gen10$ peers with 20 time steps per action.

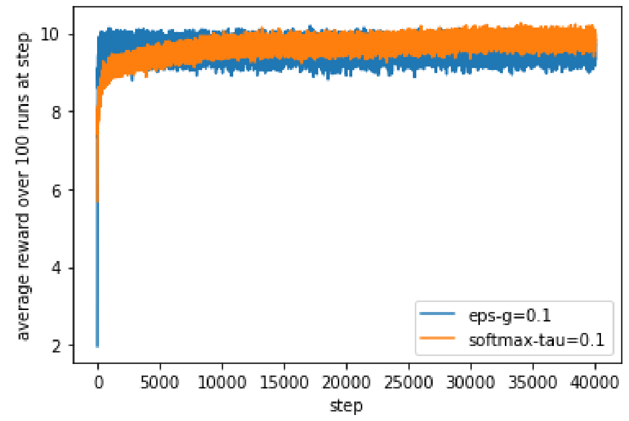


Figure 5. Average reward over time for ε-greedy and SoftMax over 40000 steps using $base5$ peers with 1 time step per action.

present noticeable improvements. We can therefore say that $\varepsilon = 0.1$ is a safe heuristic for ε-first, but also that, in this case, more peers introduces a reduction in average reward variance in the exploitation phase.

However, this reduction in average reward variance is not evidenced in Figure 8, where the variance is roughly the same for each set of peers. Regardless of this, a consistent observation from Figure 7 and Figure 8 indicates that the set with the greater number of peers (all peers in both sets are created from the same distributions) attains a higher average reward than the set with fewer peers. This is due to the fact that more peers means more chances for a single peer to be significantly better than the best peer in a set of fewer peers.

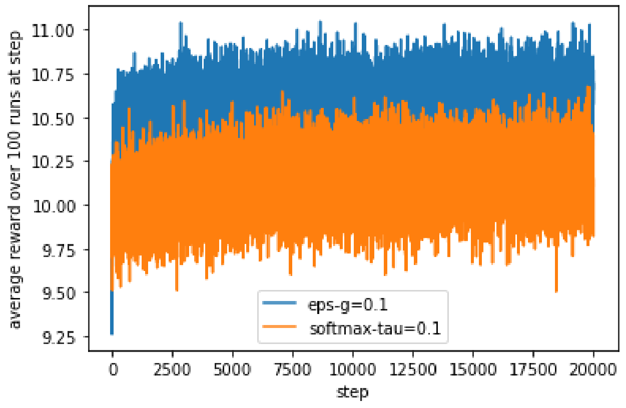


Figure 6. Average reward over time for ε-greedy and SoftMax over 20000 steps using $gen10$ peers with 1 time step per action.

6. Discussion

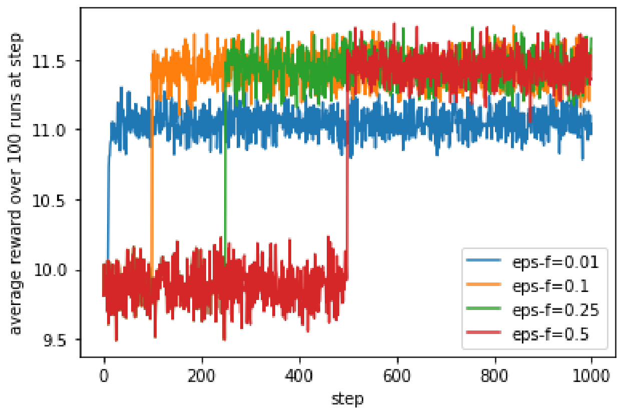
Based on our experimental results we found UCB to perform exceptionally well in each of our test scenarios. This was roughly what we had expected based on our research and findings from the related works, but were still surprised given the few amount of steps it required. We can attribute this performance to the selected C value as this controls the level of exploration and exploitation the algorithm experiences. As expected, running UCB with $C = 15$ drastically changed the behavior of the algorithm, causing the number of learning steps before stabilization to increase greatly.

Both ε-greedy and ε-first performed well but took more steps to reach a similar performance of UCB. ε-first seemed to struggle in the first $\sim$100 learning steps but quickly jumped to oscillating around the performance of UCB, while ε-greedy brought itself to under UCB more gradually.

SoftMax and Exp3 had notable short- and long-term tradeoffs when compared to the other algorithms. To better understand this we directly compared SoftMax to ε-greedy over a larger period of steps than before, and altered the number of peers as seen in Figure 5 and Figure 6. We were able to identify that, although SoftMax does start to rise above ε-greedy, this does not occur if the number of peers are increased. This gives some appeal to SoftMax if we were in a scenario where the number of peers were few and running for an extended time, but this is an unlikely situation.

7. Conclusion and Future Research

Within our peer-to-peer network environment we were able to explore a number of popular MAB algorithms, each leading to unique results. UCB in particular was the strongest of the bunch, while ε-first was slightly behind due to its startup exploration phase. Increasing the number of peers



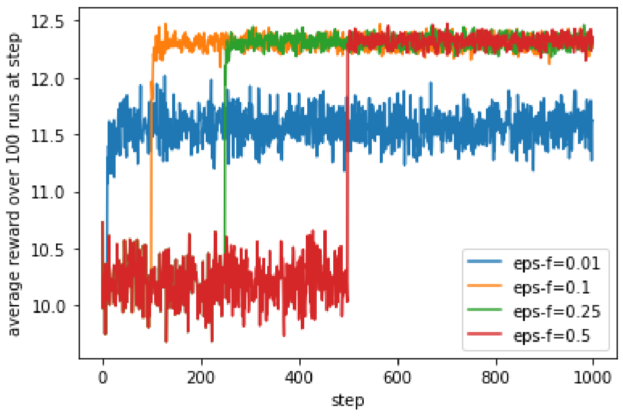


Figure 7. Comparing average reward over time for various ε values in ε-first using $gen10$ (above) and $gen20$ (below) peers with 1 time step per action.

within the network leads to an increase in the average reward value. Some negative correlation between the number of peers and the variance may exist, as seen in Figure 8, though results are inconclusive. This shows promise that our implementation may scale well as the number of peers within the network increases, however additional analysis of this variance relationship should be investigated.

This project was an exciting problem to tackle, especially given that we had no prior experience to the reinforcement learning field. Applying the teachings from our lectures and assignments allowed us to explore new and relevant algorithms, and utilize them to explore a practical research problem. Having the ability to adapt knowledge gained from a class into a tangible project is a fantastic feeling.

In the future we hope to implement the aforementioned POKER algorithm, as well as other algorithms surveyed in the related works, such as a gossip-based algorithm. The results from these future experiments would help us better understand how more complex and newer algorithms perform in comparison to these simpler ones. We would

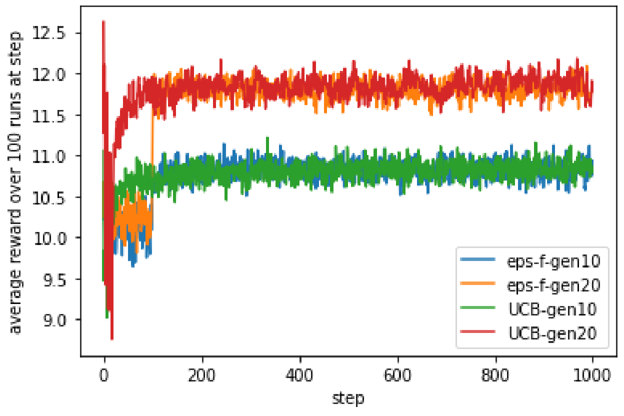


Figure 8. Comparing UCB and ε-first using $gen10$ and $gen20$ peers with 1 time step per action.

also adapt how our environment behaves, hoping to run a simulated peer-to-peer network that would allow for different forms of communication among users. This enables a more realistic stochastic environment which should make the performance comparison among algorithms a bit more interesting.

References

Avner, O. and Mannor, S. Multi-user communication networks: A coordinated multi-armed bandit approach. *IEEE/ACM Transactions on Networking*, 27(6):2192–2207, 2019. doi: 10.1109/TNET.2019.2935043.

Boldrini, S., De Nardis, L., Caso, G., Le, M. T. P., Fiorina, J., and Di Benedetto, M.-G. mumab: A multi-armed bandit model for wireless network selection. *Algorithms*, 11(2), 2018. ISSN 1999-4893. doi: 10.3390/a11020013. URL `https://www.mdpi.com/1999-4893/11/2/13`.

Hillel, E., Karnin, Z., Koren, T., Lempel, R., and Somekh, O. Distributed exploration in multi-armed bandits. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pp. 854–862, Red Hook, NY, USA, 2013. Curran Associates Inc.

Korda, N., Szörényi, B., and Li, S. Distributed clustering of linear bandits in peer to peer networks. *ArXiv*, abs/1604.07706, 2016.

Li, F., Yu, D., Yang, H., Yu, J., Karl, H., and Cheng, X. Multi-armed-bandit-based spectrum scheduling algorithms in wireless networks: A survey. *IEEE Wireless Communications*, 27(1):24–30, 2020. doi: 10.1109/MWC.001.1900280.

Modi, N., Mary, P., and Moy, C. Qos driven channel selection algorithm for cognitive radio network: Multi-user multi-armed bandit approach. *IEEE Transactions on Cognitive Communications and Networking*, 3(1):49–66, 2017. doi: 10.1109/TCCN.2017.2675901.

Schollmeier, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pp. 101–102, 2001. doi: 10.1109/P2P.2001.990434.

Si, P., Yu, F. R., Ji, H., and Leung, V. C. M. Distributed sender scheduling for multimedia transmission in wireless peer-to-peer networks. In *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*, pp. 1–5, 2008. doi: 10.1109/GLOCOM.2008.ECP.992.

Szörényi, B., Busa-Fekete, R., Hegedüs, I., Ormándi, R., Jelasity, M., and Kégl, B. Gossip-based distributed stochastic bandit algorithms. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–19–III–27. JMLR.org, 2013.

Tasoulas, Z.-G. and Anagnostopoulos, I. Improving gpu performance with a power-aware streaming multiprocessor allocation methodology. *Electronics*, 8(12), 2019. ISSN 2079-9292. doi: 10.3390/electronics8121451. URL https://www.mdpi.com/2079-9292/8/12/1451.

Vermorel, J. and Mohri, M. *Multi-armed Bandit Algorithms and Empirical Evaluation*, pp. 437–448. Machine Learning: ECML 2005. Springer Berlin Heidelberg, Berlin, Heidelberg, 1ère éd edition, 2005. ISBN 0302-9743.

Yang, M., Zhu, H., Wang, H., Koucheryavy, Y., Samouylov, K., and Qian, H. Peer to peer offloading with delayed feedback: An adversary bandit approach. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5035–5039, 2020. doi: 10.1109/ICASSP40776.2020.9053680.