Download Speed Optimization in P2P Networks Using Decision Making and Adaptive Learning

Aristeidis Karras[1](✉), Christos Karras[1], Konstantinos C. Giotopoulos[2], Ioanna Giannoukou[2], Dimitrios Tsolis[3], and Spyros Sioutas[1]

[1] Computer Engineering and Informatics Department, University of Patras, Patras, Greece
{akarras,c.karras,sioutas}@ceid.upatras.gr
[2] Department of Management Science and Technology, University of Patras, Patras, Greece
{kgiotop,igian}@upatras.gr
[3] Department of History and Archaeology, University of Patras, Patras, Greece
dtsolis@upatras.gr

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 analyzing the problem of optimizing new peer connections. In this paper, we outline the new peer scenario described as a reinforcement learning problem with MABs in order to identify the fastest peer to download from during the connection process. The MAB problem involves k slot machines which are also called one-armed bandits and pay out reward values according to an internal distribution, of which the agent is not aware. The aim is to choose a strategy to learn which arms pay out the most in order to maximize total reward over a set number of rounds. Results indicate that UCB and ε-first performed the best at selecting the optimal peer in each of our test scenarios. Contrariwise, SoftMax and ε-greedy unperformed.

Keywords: P2P networks · Reinforcement learning · Decision making · Intelligent agents · Adaptive learning · Multi-armed bandit · Strategies

1 Introduction

Peer-to-peer (P2P) computer networks provide an environment for content distribution in which the integrity of the system is not endangered by the loss of a single, centralised node. Based on [1], true p2p systems require peers to be directly accessible (without intermediary entities) and the network state or quality of service to be retained in the event that a peer is withdrawn from the network for any reason. The prerequisites for peer-to-peer networks differ per application domain. However, new peers that connect to the server for the

K. Daimi and A. Al Sadoon (Eds.): ICR 2022, AISC 1431, pp. 225–238, 2022.
https://doi.org/10.1007/978-3-031-14054-9_22

first time have no knowledge of the network status. As a result, new peers cannot be held responsible for preserving the network state and content if existing nodes disconnect. It is critical that this new peer receives relevant data as soon as feasible in order to meet both the needs of a real p2p ecosystem and any required quality of service standards. With the extra volatility of a dynamic network configuration, the velocity at which a new peer may be brought up to speed becomes significantly more important. P2P systems may also be seen as streaming applications where event detection techniques can be used to find a new peer in the network [2] while MCMC methods [5,6] and heuristics as in [3] can be used to discover the fastest peer to download from.

For example, consider the case of a p2p network wherein a new peer joins with the intent to be brought up to speed with the rest of the network as soon as feasible (i.e. download all the data in the network from other peers). However, the new peer does not have prior knowledge for the network speeds of its seeds, only just how much data it receives over time when it chooses a peer and receives data from them in 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. At this point, it is assumed that data packets are UDP datagrams.

Time steps should not be confused with steps (or rounds) in the algorithm. Multiple time steps can happen during a single round, as a step is when the agent makes a decision for a given number of time steps. Rounds and steps are used interchangeably within the paper. 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. This is named as 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 transit from one state to another at every time step. In this case, every peer is running its own Markov process in the background, irrespective of the action by the agent.

This case can be solved with a multi-armed bandit (MAB) approach where each peer is considered as an arm in the MAB algorithm. The agent will choose to pull an arm and receive reward for a certain number of time steps. During every time step, the network dynamics shift, and each arm may transit to an other 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 works [4,7,8] investigate p2p networks where the peers communicate or compete with one another. However, in our case we assume 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 MAB agents fits the learning requirements under these constrained conditions. Various algorithms are considered to solve the MAB problem, and a few are selected and implemented in order to evaluate their efficiency against this problem.

The remainder of the paper is organized as follows. In Sect. 2 related work in the field of multi-armed bandits is surveyed and covered in order to compare our

work with solutions to similar problems and verify the validity of our results. Section 3 describes the methodology in both theory and application level while Sect. 4 highlights the experimental results and their findings. A brief discussion takes place in 5 and finally, the conclusions and future directions of this work are presented in Sect. 6.

2 Related Work

Multi-armed bandits (MABs) serve as a useful abstraction for optimization problems that require decision making with reward outcomes that are initially unknown. In [9], secondary user nodes pick a single channel without knowing its quality or availability in cognitive radio networks utilizing a MAB strategy. The authors employ an upper confidence bound (UCB) algorithm named QoS-UCB. Extending the bandit model makes wireless network selection more flexible, maximizing end-user perceived quality [10]. In this formulation, the agent may perform two tasks: measure or utilise. Measuring simply permits assessment while utilising adds exploitation. Measuring is faster than utilising, which might require many stages. The payout distributions are determined by the algorithm selection. Aggressive algorithms like poker have little regret but high variability, making them less dependable [10] while more efficient methods are shown in [11].

In [4], strategies are provided for coordinating numerous MABs agents and learning stochastic network conditions. The agent transmits instead of receiving in their problem formulation. While outbound transmission speed or success may be measurable, reception rate is not. The final receiver may not have the resources to unpack transmission bundles in time, causing congestion. In [12], scheduling algorithms for MAB problems in wireless networks are presented. ε-greedy, an algorithm that balances exploration and exploitation, has flaws in its pure randomness and does not take confidence intervals into consideration. UCB uses this and gradually reduces exploration. There is a distinguish between single and multi-player multi-armed bandits (SMAB and MMAB). SMAB is used in single-peer leeching and centralised network methods while MMAB typically compromises independence for synchronisation overheads.

The study in [13] investigates how k players cooperate to locate an ε-optimal arm in an MAB environment and find communication only once players learn $\sqrt{k}$ times quicker than a single player. This strategy may be handy if network peers vary or service quality varies. Little effort has been put towards adapting bandit algorithms to p2p network contexts [7]. This paper implements the ε-greedy stochastic algorithm in a p2p network, scaling with network size, obtaining a linear speedup in terms of the number of peers, and retaining the asymptotic behaviour of the standalone version. In p2p networks, competition is unavoidable, particularly when peers attempt to stay up to date. Unpredictable competition makes management tough. Hence, distributed clustering scenarios are utilized for overcoming MAB difficulties in p2p networks [14]. All peers solve the same issue in one setting, whereas a cluster of peers solves the same problem within its cluster. This achieves an ideal regret rate, defined as the difference between the optimal reward total and the collected rewards.

In [8], an online learning strategy based on an MAB architecture is designed to handle peer rivalry and delayed feedback. The work in [15] optimises data rate transfers and minimises power consumption in wireless p2p networks. Similar constraints are common when examining graphical processing units (GPUs), as in [16]. MAB problems are also highlighted in [17], where the integration of MAB problems with recommendation systems is presented.

3 Methodology

In this section, we present a brief summary of each algorithm investigated, as well as our methodology for creating test cases and evaluating the performance of the algorithms on the new peer update task using a limited number of hyperparameters.

3.1 Employed Algorithms

The most common algorithm utilized to multi-armed bandit (MAB) problems is named ε-greedy. In ε-greedy a single hyperparameter ε is taken which indicates 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. ε-greedy is utilized as well as some of its variants in this paper.

Initially, ε-first is an ε-greedy strategy in which only exploration is done for the initial $T\varepsilon$ rounds, and pure exploitation occurs during the remaining rounds. The number of rounds (also called steps) is defined as T per run [18]. This forces exploration meaning peak rewards will be delayed, but broader experience is gained as a trade-off. Additionally, ε-decreasing is one more ε-greedy variant whereabouts the initial ε value ε_0 is decreased over the number of rounds completed. In particular, the probability of exploration at a given time t is defined as

$$\varepsilon_t = \min\left\{1, \frac{\varepsilon_0}{t}\right\} \text{ where } \varepsilon > 0 \tag{1}$$

The values for ε_0 are typically not on the interval $[0, 1]$, instead values like 1.0, 5.0, and 10.0 are used [18].

The SoftMax method (also called Boltzmann Exploration) performs action decisions based on *probability matching* methods [18]. Each arm a of k arms holds an associated probability

$$p_a = e^{Q_a/\tau} / \sum_{a'} e^{Q_{a'}/\tau} \tag{2}$$

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.

Another variant of SoftMax is Exp3, using the idea of Boltzmann Exploration and probability matching [18]. Each arm a of k arms has an associated probability

$$p_a(t) = (1-\gamma)\frac{w_a(t)}{\sum_{j=1}^{k} w_j(t)} + \frac{\gamma}{k} \tag{3}$$

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 $w_a(t)$.

In many of the previously mentioned works, UCB (Upper Confidence Bound) and slight alterations of UCB were used. Given its popularity and excellent results we hoped it would also perform well within our environment setup. Instead of selecting actions based on probabilities, UCB alters its exploration and exploitation as it gathers information about the environment. Least-taken actions are prioritized during the exploration phase, and once the estimated action values are more established, UCB exploits the action with the highest estimated reward. This action selection is derived from the following

$$A_t = argmax_a \left[Q_t(a) + C\sqrt{\log t / N_t(a)} \right] \tag{4}$$

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. The exploitation part of the equation is $Q_t(a)$, 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,

$$A_t = argmax_a \left[Q_t(a) + C\sqrt{2\log t / N_t(a)} \right] \tag{5}$$

This is a popular point of view on the original UCB algorithm that we considered to include to see how they may differ in our implementation.

Another MAB algorithm named poker was introduced in [10] which has low regret but is unreliable due to high variance. Hence, because of these problems, time constraints, and the complexity of the pseudo-code included in the paper, we decided to not cover poker in our evaluation.

3.2 Implementation

The initial peer setup for this work is comprised from a set of 5 single-state peers we call *init*5:
$PeerArm(2,1)$, $PeerArm(4,1)$, $PeerArm(6,1)$, $PeerArm(8,1)$, $PeerArm(10,1)$. This simple layout removes the simulated dynamism of multi-state peers, but provides a descent baseline evaluation for each algorithm.

For the automation of the creation of multiple peers, we introduce Algorithm 1 which takes as input a number of peers to generate, and three distributions (with their associated hyperparameters) from which it samples the number

Algorithm 1. produce_peers

Require: 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

Ensure: An array of $PeerArm$ objects Φ

Initialize Φ as an empty array
for p = 0 **to** n **do**
  $n_s \sim \mathbb{S}(s_p)$
  if $n_s < 1$ **then**
    $n_s = 1$
  end if
  Initialize Π as an empty array of means
  Initialize Σ as an empty array of standard deviations
  for p = 0 **to** n_s **do**
    $\pi \sim \mathbb{M}(m_p)$
    $\sigma \sim \mathbb{D}(d_p)$
    Append π to Π
    Append σ to Σ
  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 ϕ to Φ
end for
Return Φ

of states a peer will have, and the mean and standard deviation for each state reward. Each state requires a mean and standard deviation because rewards are generated utilizing a normal distribution.

The generate function *prod*10 set of peers utilizes Algorithm 1 to produce 10 peers using the specification below, each of which is more realistic in their transmission speed (reward) variance than *init*5. Respectively we perform the same task for 20 peers, labelled *prod*20.

- $prod10 = produce\_peers(10,$
- $np.random.poisson, dict(lam = 5.0),$
- $np.random.normal, dict(loc = 10.0, scale = 1.5),$
- $np.random.normal, dict(loc = 0.5, scale = 0.1))$

The algorithms outlined in Sect. 3.1 are implemented in *Python* 3.9 language and evaluated using a *PyCharm* Jupyter Notebook. Each algorithm utilizes a generic *BanditEnv* environment in order to execute actions and resets the environment after each execution. The estimates of the action value are computed by using a sample-average estimate of action value, with an initial estimate of 0 for each peer. This method does not exploit 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 purpose of conserving time and preserving room for future study, this work uses the sample-average estimation approach. 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. Bear in mind that the state of peer transitions will remain to be active for each time step. Therefore, the agent cannot lock the network state by taking an action for several time steps.

4 Experimental Results

Each algorithm completes 100 runs, with a fixed number of n steps. In this experiments we test for $n = 1$ and $n = 20$. 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 subsection, we present plots created by *matplotlib* library and *seaborn* package 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.

4.1 Evaluation

Figure 1 shows the general performance of each algorithm when it comes to the set of static peers, *init*5.

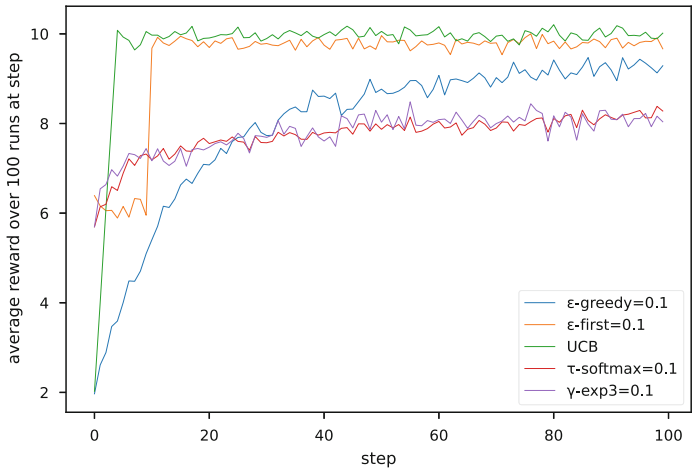

Fig. 1. Average reward over time for each algorithm over 100 steps using the *init*5 peers with 1 time step per action.

The aforementioned papers, 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 the C hyperparameter of UCB did not improve its already strong performance at $C = 1$, so we increased it instead.

Figure 2 uses the *init*5 peers with C value of UCB set to 15, and all other algorithm hyperparameters set to 0.01.

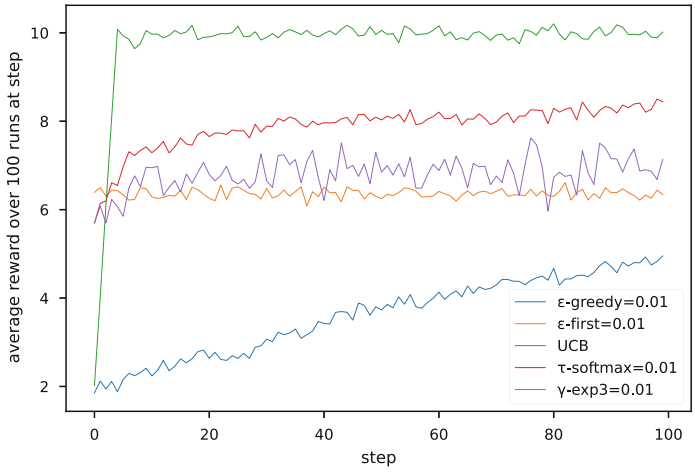

Fig. 2. Average reward over time for each algorithm over 100 steps using the *init*5 peers with 1 time step per action. Different hyperparameters are noted. UCB uses $C = 15$ as opposed to $C = 1$.

Figure 3 uses the dynamic *prod*10 peer set. Exp3 seems to peak higher than SoftMax compared to Fig. 1, but then they are almost identical. The other algorithms have similar performance, although it seems that the dominance of UCB suffers only slightly in stochastic scenarios.

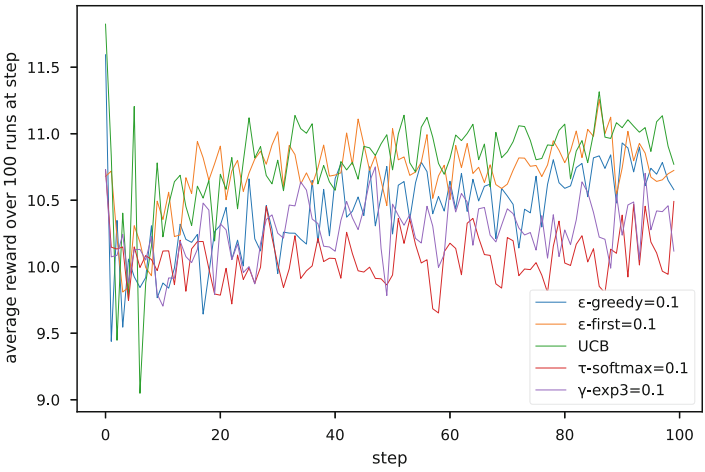

Fig. 3. Average reward over time for each algorithm over 100 steps using *prod*10 peers with 1 time step per action.

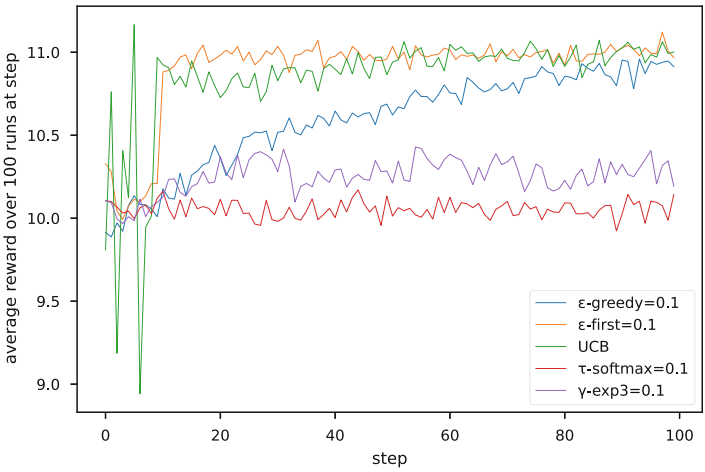

Fig. 4. Average reward over time for each algorithm over 100 steps using *prod*10 peers with 20 time steps per action.

Increasing 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 Fig. 4. However, it is also noteworthy that the overall average reward peaks higher for each algorithm, compared to Fig. 3, with 1 time step per action.

Based on the results in Fig. 1, we wanted to find out if SoftMax would ever overtake ε-greedy. In Fig. 5 we see that SoftMax eventually stabilizes with lower variance but it is still lower than ε-greedy.

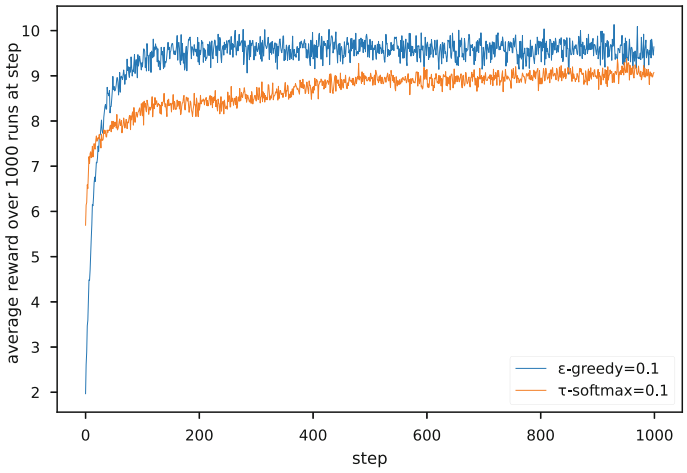

Fig. 5. Average reward over time for ε-greedy and SoftMax over 1000 steps using *init*5 peers with 1 time step per action.

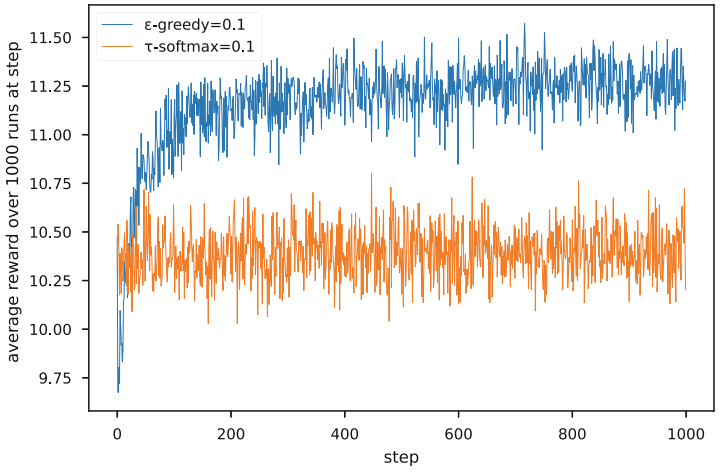

Fig. 6. Average reward over time for ε-greedy and SoftMax over 1000 steps using *prod*10 peers with 1 time step per action.

However, Fig. 6 shows that this is not the case for the more dynamic *prod*10 peers. Even after only 1000 steps, it is clear that SoftMax is not rising above ε-greedy while on *init*5 its performance was significantly higher.

As we have seen, ε-first is capable of competing with UCB in optimal peer selection. In Fig. 7, we show that extending the exploration proportion for more peers does not 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 lead to a reduction in average reward variance in the exploitation phase.

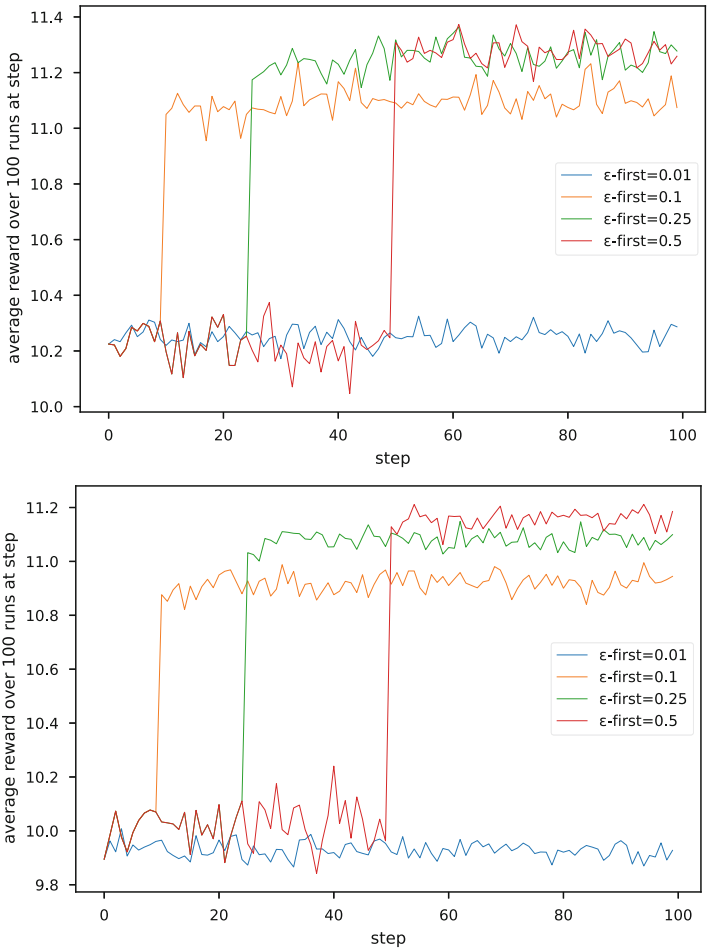

Fig. 7. Comparing average reward over time for various ε values in ε-first using 10 peers (above) and 20 peers (below) with 1 time step per action.

As seen in Fig. 7, the set with the greater number of peers, attains a higher average reward than the set with fewer peers. Note that all peers in both sets are created from the same distributions. 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. In Fig. 7 in the top subfigure, the plots are created

using the *prod*10 function, indicating that the number of peers is 10 while in the bottom subfigure, the plots are created using the *prod*20 function indicating that the number of peers is 20. Additionally, noteworthy is the fact that ε-first=0.5 took almost 50 steps to reach a descent amount of reward and therefore the performance was the highest. This leads us to the initial observation where as other works highlighted, the value of ε-first must be within $[0.01, 0.25]$ range.

5 Discussion

According to our experimental findings, UCB performed significantly well in every test situation. This was anticipated based on our study and conclusions from comparable efforts, but we were nevertheless astonished by the small number of steps required. This performance may be attributed to the specified C value, which regulates the amount that the algorithm requires of exploration and exploitation. As anticipated, running UCB with $C = 15$ substantially altered the behaviour of the algorithm, resulting in a significant increase in the number of learning steps before reaching stability.

The methods ε-greedy and ε-first performed smoothly, although they required more steps to get a robust performance. ε-first seemed to struggle in the first $\sim$10 learning steps, but it immediately leaped to fluctuating about the performance of UCB, while ε-greedy eventually got itself to beneath UCB.

In respect to the other algorithms, SoftMax and Exp3 have significant short-term and long-term trade-offs. As demonstrated in Figs. 5 and 6, we directly compared SoftMax to ε-greedy over a longer period of steps and varied the number of peers to get a better understanding. We were able to determine that SoftMax exceeds ε-greedy for low number of steps while as the number of steps increases, ε-greedy overtakes. This offers SoftMax some appeal if we were in a scenario with a small number of peers operating for a long period of time, but this is very improbable.

6 Conclusions and Future Work

In the context of the peer-to-peer network system presented, we were able to investigate a variety of well-known MAB algorithms (ε-greedy, ε-first, UCB and more) each of which produced distinct outcomes. To verify the validity of the results, related literature is surveyed in order to compare our work with solutions to similar problems. In terms of performance, the most robust of the group was UCB, while ε-first performed almost identically in the example with the 1 time step per action. Increasing the number of peers inside a network causes the average reward value to rise. There may be a negative relationship between the number of peers and the variance, but the data are ambiguous. This suggests that our solution may scale effectively as the number of peers in the network rises; nevertheless, this variance connection requires more investigation.

Future directions of this work include the implementation of the aforementioned poker algorithm along with additional methods explored in related works,

such as a gossip-based algorithm. The outcomes of these future experiments can aid in our comprehension of the performance of more complicated and modern algorithms in contrast to these undemanding ones. Moreover, a modification of the behaviour of the environment in the hopes of simulating a peer-to-peer network that would permit various types of user communication can occur. This will qualify for a more realistic stochastic environment, which should make algorithm performance comparisons more interesting and accurate, similar to a real-world scenario where noise is present to all types of communications.

References

1. 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. IEEE (2001)
2. Karras, C., Karras, A., Sioutas, S.: Pattern recognition and event detection on IoT data-streams. arXiv preprint arXiv:2203.01114 (2022)
3. Karras, C., Karras, A.: DBSOP: an efficient heuristic for speedy MCMC sampling on polytopes. arXiv preprint arXiv:2203.10916 (2022)
4. Avner, O., Mannor, S.: Multi-user communication networks: a coordinated multi-armed bandit approach. IEEE/ACM Trans. Netw. **27**(6), 2192–2207 (2019)
5. Karras, C., Karras, A., Avlonitis, M., Giannoukou, I., Sioutas, S.: Maximum likelihood estimators on MCMC sampling algorithms for decision making. In: Maglogiannis, I., Iliadis, L., Macintyre, J., Cortez, P. (eds.) AIAI 2022. IFIP Advances in Information and Communication Technology, vol. 652, pp. 345–356. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08341-9_28
6. Karras, C., Karras, A., Avlonitis, M., Sioutas, S.: An overview of MCMC methods: from theory to applications. In: Maglogiannis, I., Iliadis, L., Macintyre, J., Cortez, P. (eds.) AIAI 2022. IFIP Advances in Information and Communication Technology, vol. 652, pp. 319–332. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08341-9_26
7. Szorenyi, B., Busa-Fekete, R., Hegedus, I., Ormándi, R., Jelasity, M., Kégl, B.: Gossip-based distributed stochastic bandit algorithms. In: International Conference on Machine Learning, pp. 19–27. PMLR (2013)
8. Yang, M., Zhu, H., Wang, H., Koucheryavy, Y., Samouylov, K., 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. IEEE (2020)
9. Modi, N., Mary, P., Moy, C.: QoS driven channel selection algorithm for cognitive radio network: multi-user multi-armed bandit approach. IEEE Trans. Cogn. Commun. Netw. **3**(1), 49–66 (2017)
10. Boldrini, S., De Nardis, L., Caso, G., Le, M.T., Fiorina, J., Di Benedetto, M.G.: muMAB: a multi-armed bandit model for wireless network selection. Algorithms **11**(2), 13 (2018)
11. Vial, D., Shakkottai, S., Srikant, R.: Robust multi-agent multi-armed bandits. In: Proceedings of the Twenty-second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, pp. 161–170 (2021)

12. Li, F., Yu, D., Yang, H., Yu, J., Karl, H., Cheng, X.: Multi-armed-bandit-based spectrum scheduling algorithms in wireless networks: a survey. IEEE Wirel. Commun. **27**(1), 24–30 (2020)
13. Hillel, E., Karnin, Z.S., Koren, T., Lempel, R., Somekh, O.: Distributed exploration in multi-armed bandits. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
14. Korda, N., Szorenyi, B., Li, S.: Distributed clustering of linear bandits in peer to peer networks. In: International Conference on Machine Learning, pp. 1301–1309. PMLR (2016)
15. Si, P., Yu, F.R., Ji, H., Leung, V.C.: Distributed sender scheduling for multimedia transmission in wireless mobile peer-to-peer networks. IEEE Trans. Wirel. Commun. **8**(9), 4594–4603 (2009)
16. Tasoulas, Z.G., Anagnostopoulos, I.: Improving GPU performance with a power-aware streaming multiprocessor allocation methodology. Electronics **8**(12), 1451 (2019)
17. Silva, N., Werneck, H., Silva, T., Pereira, A.C., Rocha, L.: Multi-armed bandits in recommendation systems: a survey of the state-of-the-art and future directions. Expert Syst. Appl. **197**, 116669 (2022)
18. Vermorel, J., Mohri, M.: Multi-armed Bandit Algorithms and Empirical Evaluation. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) ECML 2005. LNCS (LNAI), vol. 3720, pp. 437–448. Springer, Heidelberg (2005). https://doi.org/10.1007/11564096_42