The Asymptotic Randomised Control Algorithm for Contextual Bandits

Samuel N. Cohen * Samuel Howard † Tanut Treetanthiploet ‡

November 28, 2021

Abstract

Work in [3] has resulted in a novel bandit algorithm based on Asymptotic Randomised Control (ARC). Here, we extend the use of the ARC algorithm to contextual bandits. The ARC algorithm involves numerically solving a fixed problem, which we improve by modifying the iteration scheme used to solve for the fixed point solution. We study the numerical performance of the ARC algorithm for contextual bandits under various scenarios using the environment provided by TensorFlow's TF-Agents library [5]. We find that the ARC algorithm algorithm performs competitively with standard methods (the Linear UCB and Linear Thompson Sampling algorithms) and significantly outperforms these methods in extreme environments.

1 Introduction

The multi-armed bandit problem is a classic problem demonstrating the trade-off between exploiting available information and exploring in order to gain more information. In the problem, an agent must sequentially choose from actions with unknown and noisy rewards, with the aim of maximising the reward over a long horizon. As rewards are received, the agent can update its beliefs about the actions and their rewards, allowing future decisions to be more informed. A bandit agent must therefore develop a strategy that balances choosing the option that is believed to be best, while also allowing sufficient exploration of the available options.

There are various bandit structures studied in the literature. In this work, we consider a contextual bandit, where at each time step we observe a 'context' upon which the action rewards depend. Based on this observed context, the agent then decides which action to choose. We focus on the available framework in the TF-Agents [5] bandits library and thus consider the following bandit setup, which unifies and generalises the two types of environment available in the library.

We consider a contextual bandit with K actions over T periods. We have an unknown (hidden) parameter θ taking values in $\mathbb{R}^d$. At each time $t \in \{1, .., T\}$, the agent observes a context vector $b_{i,t}$ (taking values in $\mathbb{R}^d$) for each action i. For ease of notation, we arrange these context vectors into a context matrix $B_t := (b_{i,t}^\top)_{i=1,...,K}$. For each action i at time t, we also have a known reward variance $\sigma_{i,t}^2$. After observing the context matrix B_t, the agent chooses one of the actions $i \in \{1, 2, ..., K\}$ and collects (and observes) the reward $R_{i,t}$, which is sampled as

$$R_{i,t} \sim N(b_{i,t}^\top \theta, \sigma_{i,t}^2).$$

*Mathematical Institute, University of Oxford and Alan Turing Institute, samuel.cohen@maths.ox.ac.uk
†Mathematical Institute, University of Oxford, samuel.howard@new.ox.ac.uk
‡Alan Turing Institute, ttreetanthiploet@turing.ac.uk

The objective of the agent is to sequentially choose the actions $(A_t)_{t=1,...,T}$ taking values in $\{1, 2, ..., K\}$ in order to maximise the expected sum of the received rewards over the total time that we play the bandit. That is, we want to maximise

$$\mathbb{E}\left[\sum_{t=1}^{T} R_{A_t,t}\right]. \tag{1}$$

In this construction of a bandit problem, the action rewards depend on the hidden parameter θ. Choosing an action and observing its reward enables us to update our beliefs about θ, and then our expected reward for future actions may also depend on the updated parameters in our estimate for θ. That is, playing an action can also improve our knowledge about different actions that are available in the future. The extent to which this occurs depends on the context vectors $b_{i,t}$, and the reward variances $\sigma^2_{i,t}$ (as, intuitively, a lower variance provides more information, which can be interpreted as variation in the amount of information gained by playing an action). It is possible that a costly action (in the sense that it has a poor reward) may provide a large amount of knowledge about the hidden parameter, so in the long term it would be beneficial to play that action early so that the subsequent choices are better than they would have otherwise been.

This paper is structured as follows. In Section 2, we formally define the contextual bandit environments available in the TF-Agents library, to which we will adapt the ARC algorithm. We also provide an overview of the Linear UCB and Linear Thompson Sampling algorithms that are currently available in this library. In Section 3, we provide an overview of the ARC algorithm, and explain how to adapt it to the contextual bandit case. We also describe how to improve the costly process of solving the fixed point problem. In Section 4, we compare the performance of the ARC algorithm with that of the Linear UCB and Linear Thompson Sampling algorithms in a variety of different contextual bandit structures. We demonstrate that the ARC algorithm performs comparably well in the standard structures, and can significantly outperform the alternatives in extreme cases with additional structure.

2 The Contextual Bandit and Benchmark Algorithms

2.1 Environments

The TF-Agents library is a TensorFlow library for bandits and reinforcement learning which offers a variety of different algorithms and environments in which to use them, along with metrics to assess their performance. We will focus on the Stationary Stochastic Environment and the Stationary Stochastic Per Arm Environment. We define these environments below, and explain how they are instances of the setup described in the introduction.

2.1.1 Stationary Stochastic Per Arm Environment

In the per-arm environment, we have a fixed hidden parameter θ taking values in $\mathbb{R}^{l+k}$, where l and k are some non-negative integers. We have two types of context: global and per-arm. At each time step, a global context vector $\tilde{b}_t$ is randomly sampled, taking values in $\mathbb{R}^l$. For each action (or 'arm') i, a per-arm context vector $\hat{b}_{i,t}$ is also sampled, each of which are in $\mathbb{R}^k$. For each action i we then concatenate the global context vector with the corresponding per-arm context for that action to make the overall context $b_{i,t}$ as follows:

$$\begin{aligned} b_{1,t} &= ([\quad \tilde{b}_t \quad], [\quad \hat{b}_{1,t} \quad]), \\ b_{2,t} &= ([\quad \tilde{b}_t \quad], [\quad \hat{b}_{2,t} \quad]), \\ &\vdots \\ b_{K,t} &= ([\quad \tilde{b}_t \quad], [\quad \hat{b}_{K,t} \quad]). \end{aligned}$$

Hence, we see that the overall context vectors $b_{i,t}$ take values in $\mathbb{R}^{l+k}$. We observe the context matrix B_t, whose rows are the vectors $b_{i,t}^\top$, before making our choice of action. We choose an action i, and then the reward $R_{i,t}$ is then sampled as

$$R_{i,t} \sim N(b_{i,t}^\top \theta, \sigma_{i,t}^2) \tag{2}$$

for some known variances $\sigma_{i,t}^2$.

In the per-arm environment, TF-Agents only supports the case where the variances $\sigma_{i,t}^2$ are constant for all actions i and all time t. We later introduce the scenario in which the variances $\sigma_{i,t}^2$ are a known function of the context vectors $b_{i,t}$ (see Section 2.1.3).

To obtain the greatest reward at a given time t, by looking at (2) we see that we want to choose the action whose concatenated context vector $b_{i,t}$ has the greatest dot product with the hidden parameter θ. Note that in this case, playing an action and observing the resulting reward provides information about the hidden parameter θ. Other future actions may have rewards dependent on the updated parameters in θ, so playing an action can improve our knowledge of the reward of other actions in the future.

2.1.2 Stationary Stochastic Environment

The TF-Agents library also has a Stationary Stochastic Environment, in which the rewards for each action are independent. We now describe how this structure fits into the setup described in the introduction.

In this environment, for each action i we have a fixed hidden parameter, say $\theta_i \in \mathbb{R}^l$ for some dimension l. The parameters θ_i are independent. There is a context generating function, which at each time t randomly generates a context vector $\tilde{b}_t$ taking values in $\mathbb{R}^l$. Note that we observe $\tilde{b}_t$ before making our choice of action. We choose an action i, and then the reward R_t is sampled as

$$R_{i,t} \sim N(\tilde{b}_t^\top \theta_i, \sigma_{i,t}^2) \tag{3}$$

for some variance $\sigma_{i,t}^2$. Hence, at each time t we would like to choose the action whose hidden parameter has the greatest dot product with the current context $\tilde{b}_t$.

To see that this is an instance of the bandit structure defined in the introduction, set the hidden parameter $\theta = (\theta_1, ..., \theta_K)$ to be the concatenation of the individual hidden parameters for the actions $i \in \{1, ..., K\}$. We set B_t to be the block matrix consisting of rows that are the concatenation of zero vectors and the context vector $\tilde{b}_t$ in the following manner:

$$\begin{pmatrix} \begin{bmatrix} & \tilde{b}_t^\top & \end{bmatrix} & 0 & \dots & 0 \\ 0 & \begin{bmatrix} & \tilde{b}_t^\top & \end{bmatrix} & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & \begin{bmatrix} & \tilde{b}_t^\top & \end{bmatrix} \end{pmatrix}. \tag{4}$$

We choose an action i based off our current beliefs, and then the reward $R_{i,t}$ is then sampled as

$$R_{i,t} \sim N(b_{i,t}^\top \theta, \sigma_{i,t}^2) \tag{5}$$

for some known variances $\sigma_{i,t}^2$. Here, the $\sigma_{i,t}^2$ are as described in the introduction, and $b_{i,t}^\top$ is the ith row of the above matrix. This is now an instance of the bandit setup described in the introduction. In the Stationary Stochastic Environment, TF-Agents supports the case where the variances $\sigma_{i,t}^2$ can be different for different actions i, but do not depend on the time t.

Note that in this case, playing an action and observing the resulting reward only gives us information about the hidden parameter of the chosen action, so the actions have independent rewards (in contrast to the Stationary Stochastic Per Arm Environment). The block structures of these parameters enable the actions to behave independently as required in the Stationary Stochastic Environment, while being an instance of the general bandit setup considered.

2.1.3 Per Arm Environment with Changing Variance

We also consider a new type of bandit environment, which is a generalisation of the Stationary Stochastic Per Arm Environment described earlier. In this new environment we consider the case where the variance $\sigma^2_{i,t}$ of the sampled reward is a function of the context of the action that is chosen, and this variance function $\sigma^2_{i,t} = \sigma^2(b_{i,t})$ is known to the agent. This is similar to the stochastic bandit with heteroscedastic noise considered by Kirschner and Krause in [7], but here we consider the contextual bandit case. Note that this is just a generalisation of the Stochastic Linear Per Arm Environment already implemented in TF-Agents that was described in Section 2.1.1, where the variance was the same for all the rewards.

Intuitively, an action with lower variance in the reward provides more information, and it can be beneficial to play a costly yet informative action early so that future decisions are more informed. For instance, a reward sampled as $R_{i,t} \sim N(b_{i,t}^\top \theta, 1)$ should be preferred to a reward sampled as $R_{i,t} \sim N(b_{i,t}^\top \theta, 2)$ (assuming the same context $b_{i,t}$ in both cases), as the smaller variance ensures a better estimate of the mean $b_{i,t}^\top \theta$, thus enabling improved updating of the estimate for the hidden parameter θ. This can be seen in the parameter updates formulae (6) and (7) used in the next section.

The ARC algorithm considers the learning value of an action, so can exploit this additional information both in exploration and in the estimation of the parameters. On the other hand, the Linear UCB and Linear Thompson Sampling algorithms do not consider this when deciding how to explore, hence only benefit from this information when updating the estimation parameters. For a further discussion of this idea, see Section 2 of [12], and Example 1 in [7].

2.2 Existing algorithms

Two of the main algorithms in the TF-Agents bandits suite are the Linear UCB [9] and Linear Thompson Sampling [1] algorithms. These are applicable to the environments described in Section 2.1 because the expectation of the reward function is linear in θ. We will focus on these two algorithms in our analysis.

In the descriptions of the algorithms below, we use the parameters M_t and Σ_t that we later use in the ARC algorithm (see Section 3.1). These should be interpreted as the (posterior) mean and variance of the hidden parameter θ at time t respectively.

We let d denote the dimension of the hidden parameter θ and the contexts $b_{i,t}$. If we are in the Stationary Stochastic Per Arm Environment with the global contexts taking values in $\mathbb{R}^l$ and the per-arm contexts taking values in $\mathbb{R}^k$, this means that we have $d = l + k$. If instead we are in the Stationary Stochastic Environment with the contexts $\tilde{b}_t$ taking values in $\mathbb{R}^l$, then we have $d = Kl$.

2.2.1 Parameter updates

In each of the algorithms that we consider, we represent the state of the system with a Markov process (M_t, Σ_t). We should interpret M_t and Σ_t as the (posterior) mean and variance of the hidden parameter θ at time t respectively. Note that this parameterisation is used in the ARC literature, so we adapt the pseudocode for Linear UCB [9] and Linear Thompson Sampling [1] below to use these parameters for consistency.

We use and update the parameters as follows. Let $\mathcal{G}_t$ denote our historical observation at time t. We assume that $\mathcal{L}(\theta|\mathcal{G}_{t-1}) = N(M_{t-1}, \Sigma_{t-1})$. At time t, the agent observes the context $B_t := (b_{i,t})$ and chooses the option A_t. Suppose that the agent chooses $A_t = i$, and consequently observes and collects the reward $R_{i,t} \sim N(b_{i,t}^\top \theta, \sigma^2_{i,t})$. Viewed as a Kalman filter problem ([2],[6]),

we obtain $\mathcal{L}(\theta|\mathcal{G}_t) = \mathcal{L}(\theta|\mathcal{G}_t, B_t, A_t, R_{A_t,t}) = N(M_t, \Sigma_t)$ where we use the updates

$$M_t \leftarrow \Big(\Sigma_{t-1}^{-1} + \frac{b_{i,t}b_{i,t}^\top}{\sigma_{i,t}^2}\Big)^{-1}\Big(\Sigma_{t-1}^{-1}M_{t-1} + \frac{R_{i,t}b_{i,t}}{\sigma_{i,t}^2}\Big), \tag{6}$$

$$\Sigma_t \leftarrow \Big(\Sigma_{t-1}^{-1} + \frac{b_{i,t}b_{i,t}^\top}{\sigma_{i,t}^2}\Big)^{-1}. \tag{7}$$

2.2.2 The Linear UCB algorithm

The Upper Confidence Bound family of algorithms are designed to be 'optimistic in the face of uncertainty'. To encourage exploration, 'optimistic estimates' are calculated for each action, in which an additional term is added to the current expected reward estimates, based on the confidence in these estimates. The scaling of this additional term is determined by the standard deviation in the reward estimate, resulting in an upper confidence bound for the reward for each action. The algorithm then chooses the action with the largest 'optimistic estimate'.

The Linear UCB algorithm implemented in the TF-Agents library is based on the algorithm in [9]. We formally describe the Linear UCB algorithm below. The variable ρ is a hyperparameter set by the user to determine the level of exploration (the default value is $\rho = 1$).

Algorithm 1: LinUCB Algorithm

Input: ρ, T
$m \leftarrow \mathbf{0}_{d\times 1}$ (the zero vector of length d);
$\Sigma \leftarrow \mathbf{I}_d$;
for $t = 1, ..., T$ **do**
  for $i \in \{1, 2, ..., K\}$ **do**
    $p_{i,t} \leftarrow m^\top b_{i,t} + \rho\sqrt{b_{i,t}^\top \Sigma b_{i,t}}$;
  end
  Choose the action $a_t = \text{argmax}_{i\in\{1,2,...,K\}}\ p_{i,t}$ (with ties broken arbitrarily);
  Collect the reward $R_{a_t,t}$;
  Update the parameters according to the formulae (6) and (7).
end

2.2.3 The Linear Thompson Sampling algorithm

Thompson Sampling is a family of Bayesian algorithms which sample the chosen action according to the posterior probability of that action being the best choice. The Linear Thompson Sampling algorithm implemented in the TF-Agents library is based on the algorithm in [1] (specifically Algorithm 3 in the supplementary material). This particular implementation of Thompson Sampling randomly samples a value for each action according to the current posterior distribution for that action's reward, and then chooses the action whose sampled value is largest.

We formally describe the Linear Thompson Sampling algorithm below. The variable ρ is a hyperparameter set by the user, which determines the level of exploration by scaling the size of the posterior variance (the default value is $\rho = 1$).

Algorithm 2: Linear Thompson Sampling algorithm

Input: ρ, T
$m \leftarrow \mathbf{0}_{d\times 1}$ (the zero vector of length d);
$\Sigma \leftarrow \mathbf{I}_d$;
for $t = 1, ..., T$ **do**
  for $i \in \{1, 2, ..., K\}$ **do**
    $\mu_i \leftarrow m^\top b_{i,t}$;
    $\sigma_i^2 \leftarrow \rho^2 b_{i,t}^\top (D^\top D + \mathbf{I}_{l+k})^{-1} b_{i,t}$;
    Sample $p_{i,t}$ as

$$p_{i,t} \sim N(\mu_i, \sigma_i^2); \tag{8}$$

  end
  Choose the action $a_t = \text{argmax}_{i\in\{1,2,...,K\}}\ p_{i,t}$ (with ties broken arbitrarily);
  Collect the reward $R_{a_t,t}$;
  Update the parameters according to the formulae (6) and (7).
end

3 The ARC algorithm for the Contextual Bandit

3.1 General description of the ARC algorithm for the Contextual Bandit

The ARC algorithm [3] is a general approach to give an approximate solution to a stochastic control problem with small noise. We often find this scenario when we model the unknown parameter θ as a hidden random variable and infer its value using the filtering theorem.

In this work, we extend the use of the ARC algorithm to the contextual framework. Note that in the bandit setup used in [3], the parameters for each action are fixed throughout the game, whereas now that we consider the contextual bandit, the action contexts $b_{i,t}$ change at each time step. We will apply the ARC algorithm as usual, but with the time-dependent context matrix B_t in place of the fixed parameter matrix used in the original paper. The ARC algorithm uses a learning function L_t^λ (defined below) which is dependent on the reward function that is being used. We must therefore re-derive the expression for L_t^λ for the linear reward function used in our setup.

The ARC algorithm considers a function that represents the value of playing each action. This incorporates both the instantaneous reward of that action and the amount of information gained. A second order Taylor expansion of a smooth approximation to this function is then constructed, which provides an approximation to the value function that can be computed. Using these approximations, a discounted recursion formula is constructed to approximate the value function over the remaining time we can play the bandit, and the solution to the resulting fixed point equation is used to determine which action is chosen. For details, see [3].

To give a precise definition of how we use the ARC algorithm in our setup, we define the conditional expectation

$$f_t^{(i)}(m, \Sigma, B_t) := \mathbb{E}[R_{i,t} | M_{t-1} = m, \Sigma_{t-1} = \Sigma, B_t]. \tag{9}$$

Here, $f_t(m, \Sigma)$ is a vector of the expected rewards, conditional on our current knowledge.

We define the information propagation of the ith option by

$$\mu_t^{(i)}(m, \Sigma) := \mathbb{E}[M_t - M_{t-1} | M_{t-1} = m, \Sigma_{t-1} = \Sigma, A_t = i, B_t], \tag{10}$$

$$(\sigma\sigma^\top)_t^{(i)}(m, \Sigma) := \text{Var}[M_t - M_{t-1} | M_{t-1} = m, \Sigma_{t-1} = \Sigma, A_t = i, B_t], \tag{11}$$

$$b_t^{(i)}(m, \Sigma) := \mathbb{E}[\Sigma_t - \Sigma_{t-1} | M_{t-1} = m, \Sigma_{t-1} = \Sigma, A_t = i, B_t]. \tag{12}$$

We now define the learning function used for the ARC algorithm, whose terms arise from the second order Taylor expansion of the approximated value function. Extending the derivation in [3], for a fixed $\lambda > 0$ we define $L_t^\lambda : \mathbb{R}^K \times \mathbb{R}^d \times \mathbb{S}_+^d \to \mathbb{R}^K$ at time t by

$$L_{i,t}^\lambda(a, m, \Sigma, B_t) := \langle \mathcal{B}_t^\lambda(a, m, \Sigma); b_t^{(i)}(m, \Sigma)\rangle + \langle \mathcal{M}_t^\lambda(a, m, \Sigma); \mu_t^{(i)}(m, \Sigma)\rangle + \frac{1}{2}\langle \Xi_t^\lambda(a, m, \Sigma); (\sigma\sigma^\top)_t^{(i)}(m, \Sigma)\rangle, \quad (13)$$

where $\mathcal{B}^\lambda$, $\mathcal{M}^\lambda$ and Ξ^λ are given by

$$\begin{aligned}
\mathcal{B}_t^\lambda &:= \sum_i \nu_i^\lambda(a)(\partial_\Sigma f_t^{(i)}),\\
\mathcal{M}_t^\lambda &:= \sum_i \nu_i^\lambda(a)(\partial_m f_t^{(i)}),\\
\Xi_t^\lambda &:= \sum_i \nu_i^\lambda(a)(\partial_m^2 f_t^{(i)}) + \frac{1}{\lambda}\sum_{j,i} \eta_{ji}^\lambda(a)(\partial_m f_t^{(j)})(\partial_m f_t^{(i)})^\top,
\end{aligned}$$

where we define

$$\nu_i^\lambda(a) := \frac{\exp(a_i/\lambda)}{\sum_{j=1}^K \exp(a_j/\lambda)} \qquad \text{and} \qquad \eta_{ji}^\lambda(a) := \nu_j^\lambda(a)\big(\mathbb{I}(j = i) - \nu_i^\lambda(a)\big). \quad (14)$$

The 'incremental reward' function $\alpha_t^\lambda(m, \Sigma)$ is then defined to be a solution a to the fixed point problem

$$a = f_t(m, \Sigma, B_t) + \Big(\frac{\beta}{1-\beta}\Big) L_t^\lambda(a, m, \Sigma, B_t),$$

where $\beta \in [0, 1)$ is a hyperparameter chosen by the user. Following the heuristic suggestion in [3] inspired by Russo [11], we will set $\beta = 1 - \frac{1}{T}$ in our experiments. This fixed point problem is solved numerically (see Section 3.2 for details). We can interpret the ith component of $\alpha_t(m, \Sigma)$ as the 'incremental reward' of the ith action when θ is described by the posterior parameter (m, Σ). This incremental reward can decomposed into two components corresponding to exploitation (the f_t term) and exploration (the $L_{i,t}^\lambda$ term) under the learning environment, as seen by the fixed point equation above.

To choose an action, the ARC algorithm then samples from the softmax function

$$\nu_i^\lambda(a) := \frac{\exp(a_i/\lambda)}{\sum_{j=1}^K \exp(a_j/\lambda)} \quad (15)$$

of the 'incremental reward' $\alpha_t(m, \Sigma)$, so $\nu_i^\lambda(\alpha_t)$ takes values in $\mathbb{R}^K$. The softmax function $\nu_i^\lambda(a)$ can be thought of as a smooth approximation of the argmax function. Hence, roughly speaking, we are likely choose the maximum component of α_t.

In order to ensure the appropriate exploration-exploitation trade-off, we choose the parameter λ to be $\lambda = \rho|\Sigma|$, where $\rho > 0$ is a hyperparameter chosen by the user that determines the level of exploration. In fact, one can generalise the ARC algorithm to use a more general choice of λ, or replace the softmax function (15) by an alternative (see [3] for details).

In our bandit setup, the reward is linear in θ. Therefore, we can evaluate f_t and L_t^λ explicitly in this case to give the following formulae (see Appendix A for details).

$$f_t(m, \Sigma, B_t) = B_t m, \quad (16)$$

$$L_{j,t}^\lambda(a, m, \Sigma, B_t) = \frac{1}{2\lambda}\left(\frac{1}{s_{jj} + P_j^{-1}}\right)\left(\sum_{i=1}^K \nu_i^\lambda(a) s_{ij}^2 - \left(\sum_{i=1}^K \nu_i^\lambda(a) s_{ij}\right)^2\right), \quad (17)$$

where $B_t = [b_{1,t}, b_{2,t}, ..., b_{K,t}]^\top$ and $s_{ij} = s_{ij}(\Sigma, B_t)$ is defined as

$$s_{ij}(\Sigma, B_t) = b_{i,t}^\top \Sigma\, b_{j,t}. \quad (18)$$

The ARC algorithm for contextual bandits can thus be described as follows.

Algorithm 3: ARC Algorithm

Input: ρ, β, T
$m \leftarrow \mathbf{0}_{d\times 1}$ (the zero vector of length d);
$\Sigma \leftarrow \mathbf{I}_d$;
Let f_t, L_t^λ be defined as in (16) and (17);
for $t = 1, ..., T$ **do**
  Set $\lambda = \rho|\Sigma|$;
  Numerically solve the fixed point equation

$$\alpha = f_t(m, \Sigma, B_t) + \Big(\frac{\beta}{1-\beta}\Big) L_t^\lambda(\alpha, m, \Sigma, B_t); \tag{19}$$

  Sample the action a_t with $\mathbb{P}(a_t = i) = \nu_i^\lambda(a)$;
  Collect the reward $R_{a_t,t}$;
  Update the parameters according to the formulae (6) and (7).
end

3.2 Accelerating the fixed point calculation

Recall that in the ARC algorithm, we need to numerically solve the fixed point problem

$$a = f_t(m, \Sigma, B_t) + \Big(\frac{\beta}{1-\beta}\Big) L_t^\lambda(a, m, \Sigma, B_t). \tag{20}$$

This calculation may be computationally costly, so we investigate different methods to solve for the fixed point solution in order to accelerate the calculation.

We use `scipy.optimize.root` to solve this fixed point problem numerically. This solver offers a selection of methods to use to find the solution. The default is to use the Powell hybrid method as implemented in [10]; information regarding the other methods is available in the SciPy [13] documentation.

We run 50 simulations of a 10-armed bandit structure with horizon $T = 100$, where in each simulation we generate a new hidden parameter θ taking values in $\mathbb{R}^{10}$, with each entry sampled from an $N(0, 1)$ distribution. Our contexts $b_{i,t}$ are uniformly sampled as rows from the matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

At each fixed point calculation, we use each of the available numerical methods and record the time taken to find the solution. For each method, we take the mean of the recorded times taken at each time step over the 50 simulations, and then plot the cumulative means to obtain the following plot.

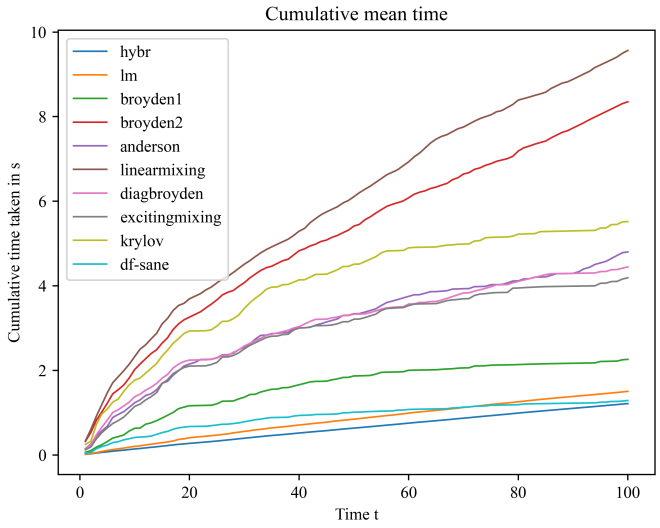

Figure 1: Plot of cumulative time of the different numerical methods

We see that in general, the Powell hybrid and Levenberg-Marquardt (denoted 'lm' in Figure 1) methods have a low cumulative time. However, their plots are linear and in the later time steps have a steeper gradient than some of the alternative methods. This is to be expected as the rate of convergence of the other methods should be very fast once we are within their required radius of convergence, but are very slow otherwise (which often occurs towards the start when Σ is large, leading to the curves seen above).

We obtain a better picture if we take the mean over only the values less than 0.1s.

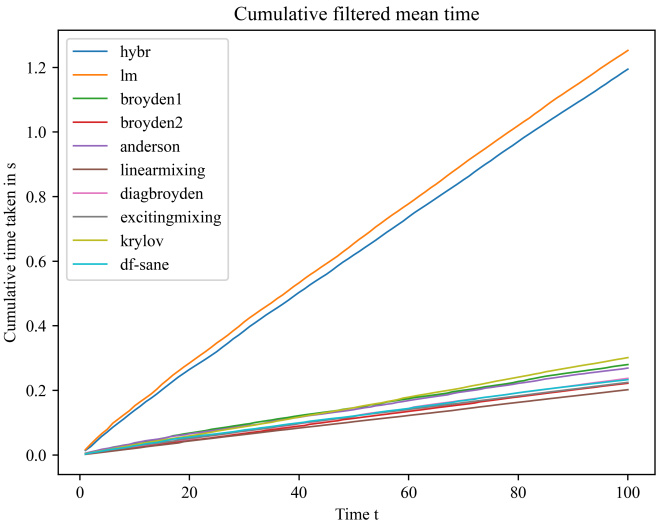

Figure 2: Plot of filtered cumulative time of the different numerical methods

Here we see that when we are in the case of rapid convergence, the Powell hybrid and Levenberg-Marquardt methods are significantly slower than the alternatives. Plotting only the alternatives, we obtain the following plot.

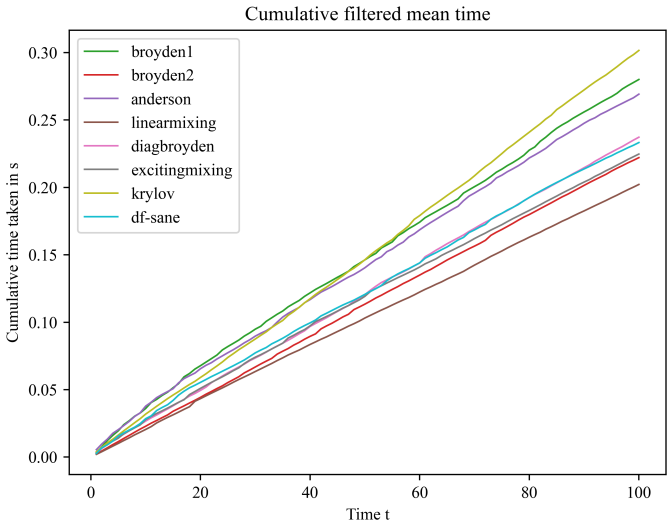

Figure 3: Plot of filtered cumulative time of the different numerical methods

We see that of the alternative methods, 'linear mixing' is generally the fastest in the cases when it converges rapidly, although there is little difference between many of the methods. If we consider how frequently these alternative methods fail to converge quickly, we see that this is most rare for the derivative free method 'df-sane' from [4] (as evidenced by its significantly lower cumulative time in Figure 1). Using the method 'df-sane' appears to offer the best compromise between the maximum speed of convergence, and being in the region of rapid convergence as often as possible.

Therefore in our implementation we will first run 10 iterations of the 'df-sane' method. If at this point we have not converged to a solution then this suggests we are not in the required radius of convergence, so we instead use the Powell hybrid method to increase the likelihood of convergence while retaining relatively quick computation. We make up to three attempts to solve using the the hybrid method, using different starting points for the iterations each time. This results in significantly faster computation of the fixed point a on average compared to using the default method, while avoiding any extremely slow calculations. If the algorithm fails to converge to a solution in all of these attempts to numerically solve the fixed point problem, then we instead use the Linear UCB policy as described earlier to make the decision. This is summarised below. We use the default tolerances to determine if convergence to a solution occurs; see the SciPy [13] documentation for details.

Algorithm 4: Method to solve fixed point problem

Perform 'df-sane' for 10 iterations, with initial iteration $f + (\frac{\beta}{1-\beta})L^{\lambda}(f, m, \Sigma)$;
if *Convergence to a solution has occured* **then**
  end
Perform 'hybrid' with initial iteration $f + (\frac{\beta}{1-\beta})L^{\lambda}(f, m, \Sigma)$;
if *Convergence to a solution has occured* **then**
  end
Perform 'hybrid' with initial iteration $f + (\frac{\beta}{1-\beta})L^{\lambda}(P, m, \Sigma)$, where $P = (\sigma_{1,t}^{-2}, ..., \sigma_{K,t}^{-2})$ is the vector of inverse variances;
if *Convergence to a solution has occured* **then**
  end
Perform 'hybrid' with initial iteration $f + (\frac{\beta}{1-\beta})L^{\lambda}(\mathbf{1}, m, \Sigma)$, where $\mathbf{1} = (1, ..., 1) \in \mathbb{R}^k$ is the ones vector of length K;
if *Convergence to a solution has occured* **then**
  end
else
  Take Linear UCB decision.
end

4 Experimental Performance

We will use the TF-Agents library to assess the performance of the algorithms in a variety of different contextual bandit structures. As is standard in the bandit literature (see [8]), we use regret as a metric to analyse algorithm performance. Regret is defined as the difference between the sum of the expected rewards of an 'oracle' agent that chooses the optimum action at each time step, and the sum of the reward received by the agent that we are assessing. In our bandit setup, that is

$$\text{Regret}(T) := \sum_{t=1}^{T} \left(b_{a_t^*,t}^\top \theta - R_{a_t,t} \right) \tag{21}$$

where (a_t) is the sequence of actions chosen by the algorithm, and (a_t^*) is the sequence of optimal actions chosen by the 'oracle' agent. To assess the performance of the algorithms, we will plot the regret achieved by each algorithm against time. We will also plot the cumulative number of suboptimal arms that the algorithms choose against time, to see how quickly they locate the optimal actions.

The TF-Agents bandits suite currently supports the metrics 'RegretMetric', 'SuboptimalArmsMetric', 'ConstraintViolationsMetric' and 'DistanceFromGreedyMetric'. To remove the effect of the random noise in the rewards on our regret plots, for this project a new pseudo-regret metric 'ExpectedRegretMetric' was added. This metric computes the expected regret of the choice of action at each time step, which is calculated using the expected value of the chosen reward rather than using the received reward itself. That is, using the same notation as above,

$$\text{ExpectedRegret}(T) := \sum_{t=1}^{T} \left(b_{a_t^*,t}^\top \theta - b_{a_t,t}^\top \theta \right). \tag{22}$$

The 'SuboptimalArmsMetric' was also altered to support cases in the per-arm environment where two actions have the same contexts, in order to treat such actions identically. We will use these two metrics when assessing the performance of the algorithms below, and in the following examples we use 'regret' to refer to the expected regret described here.

We now compare the performance of the Linear UCB, Linear Thompson Sampling, and ARC algorithms on the environments we have previously described. Due to computational restrictions, we run four different exploratory parameters for each algorithm, so better optimisation of the exploratory parameters is possible. Nevertheless we can still compare the performance of the algorithms in these examples.

4.1 Stochastic Linear Environment

We run 1000 simulations of a 6-armed bandit with the dimension of the parameters θ_i and $\tilde{b}_t$ set to $l = 5$. Therefore the concatenated hidden parameter θ lies in $\mathbb{R}^{30}$, and we generate a new θ for each simulation. Each entry of the context $\tilde{b}_t$ and the hidden parameters θ_i are uniformly sampled as an integer in the range [-10, 10]. The variance in the reward is $\sigma_{i,t} = 10$ each time, for all actions i. We take the horizon to be $T = 200$. As explained earlier, we set $\beta = 1 - \frac{1}{T} = 0.995$ for the ARC algorithm. We set the exploration parameters ρ to be 1.5, 2, 5 and 10 for Linear UCB, and 1, 1.5, 2 and 5 for Linear Thompson Sampling. For ARC we set ρ to be 0.01, 0.1, 1 and 10. We obtain the following expected regret and suboptimal arm plots.

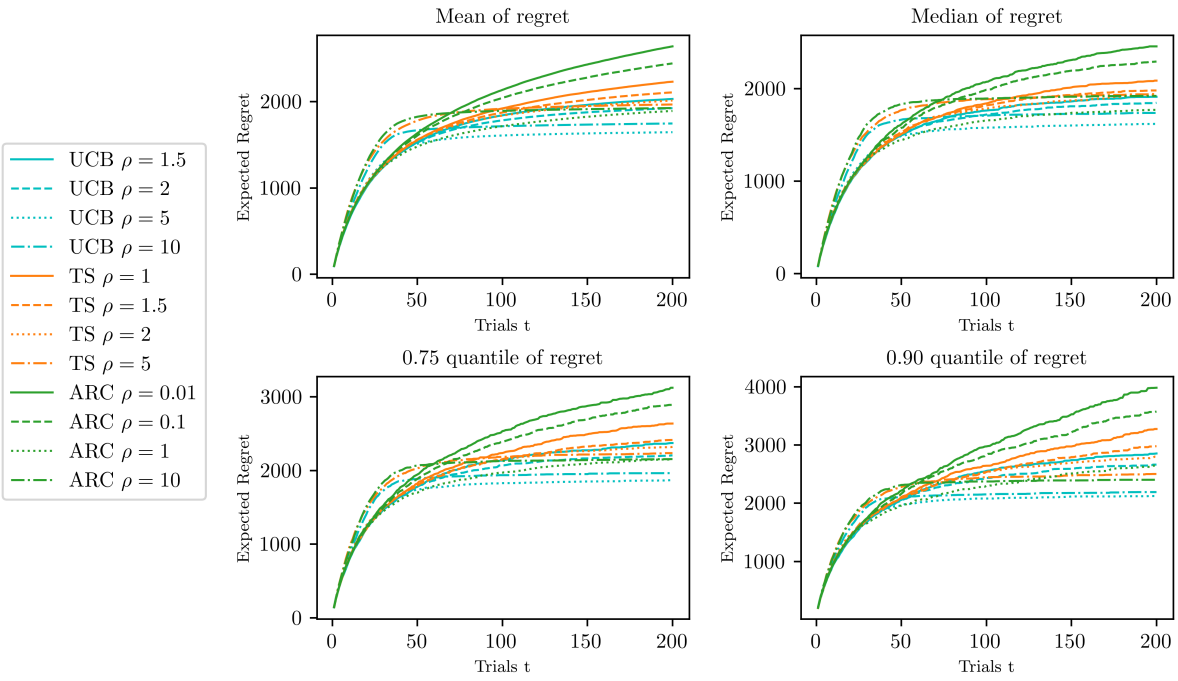

Figure 4: Plot of the cumulative regret for the different bandit algorithms in the Stochastic Linear Environment

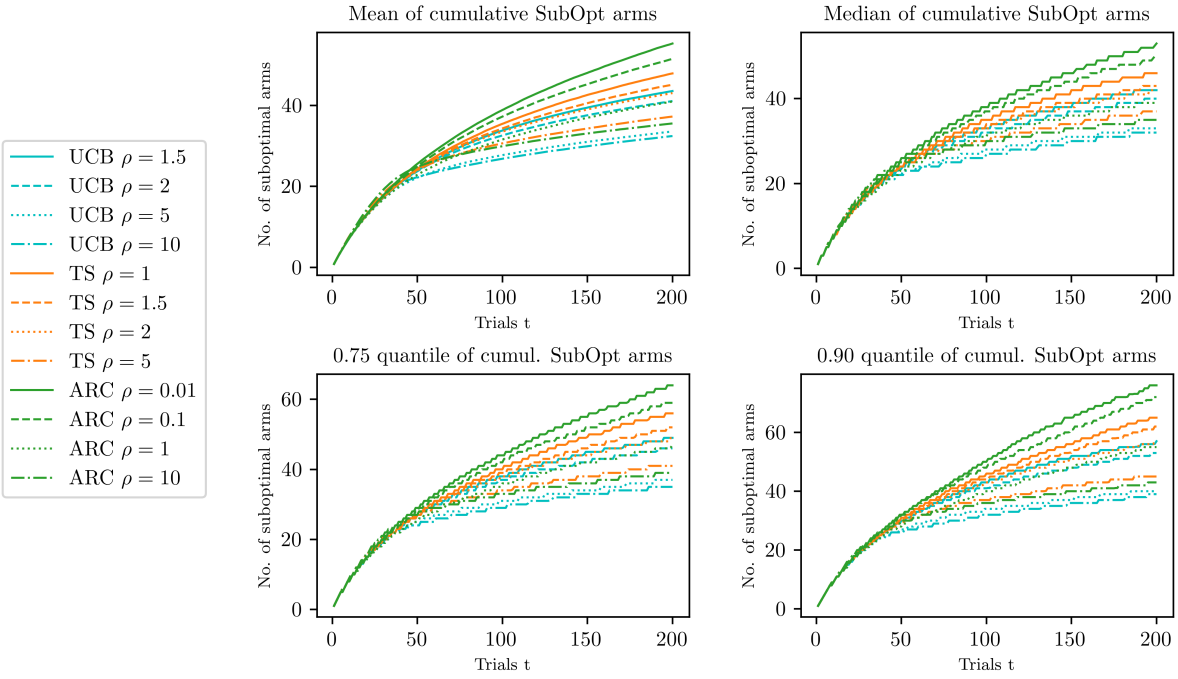

Figure 5: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Stochastic Linear Environment

We see that the algorithms all quite similarly when a good exploratory parameter is chosen. Taking $\rho = 5$ works best for Linear UCB, and performs slightly better than the best performing Linear Thompson

Sampling and ARC algorithms. Taking $\rho = 1$ provides the lowest regret on average for the ARC algorithm and provides low regret quantiles. Setting $\rho = 10$ for the ARC algorithm leads to high exploration at the start which increases the regret at first, but the algorithm subsequently makes better choices later on as can be seen in the suboptimal arms plot (Figure 5) and by the end of the game also achieves low regret. In this example the actions are independent and there is no additional structure for the algorithms to exploit, but ARC still performs relatively well.

4.2 Stochastic Linear Per Arm Environment

We run 1000 simulations of a 20-armed bandit with the dimension of the global context set to $l = 10$ and the dimension of the per-arm context set to $k = 20$. Therefore the hidden parameter θ lies in $\mathbb{R}^{30}$, and we generate a new θ for each simulation. Each entry of the hidden parameter θ and the contexts $b_{i,t}$ is uniformly sampled as an integer in the range [-10, 10]. The variance in the reward is 20 each time. We take the horizon to be $T = 100$ and therefore set $\beta = 1 - \frac{1}{T} = 0.99$ for the ARC algorithm accordingly as explained earlier. We set the exploration parameters ρ to be 0.1, 0.5, 1 and 1.5 for both Linear UCB and Linear Thompson Sampling, and ρ to be 0.01, 0.1, 1 and 10 for ARC. We obtain the following expected regret and suboptimal arm plots.

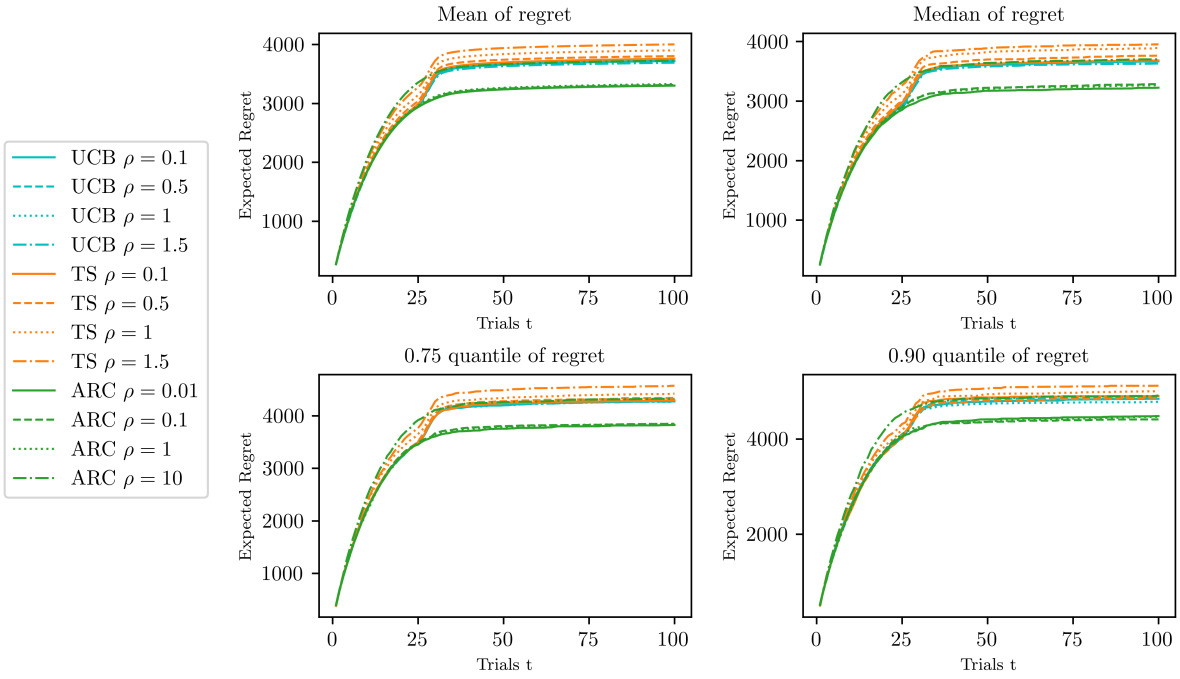

Figure 6: Plot of the cumulative regret for the different bandit algorithms in the Stochastic Linear Per Arm Environment

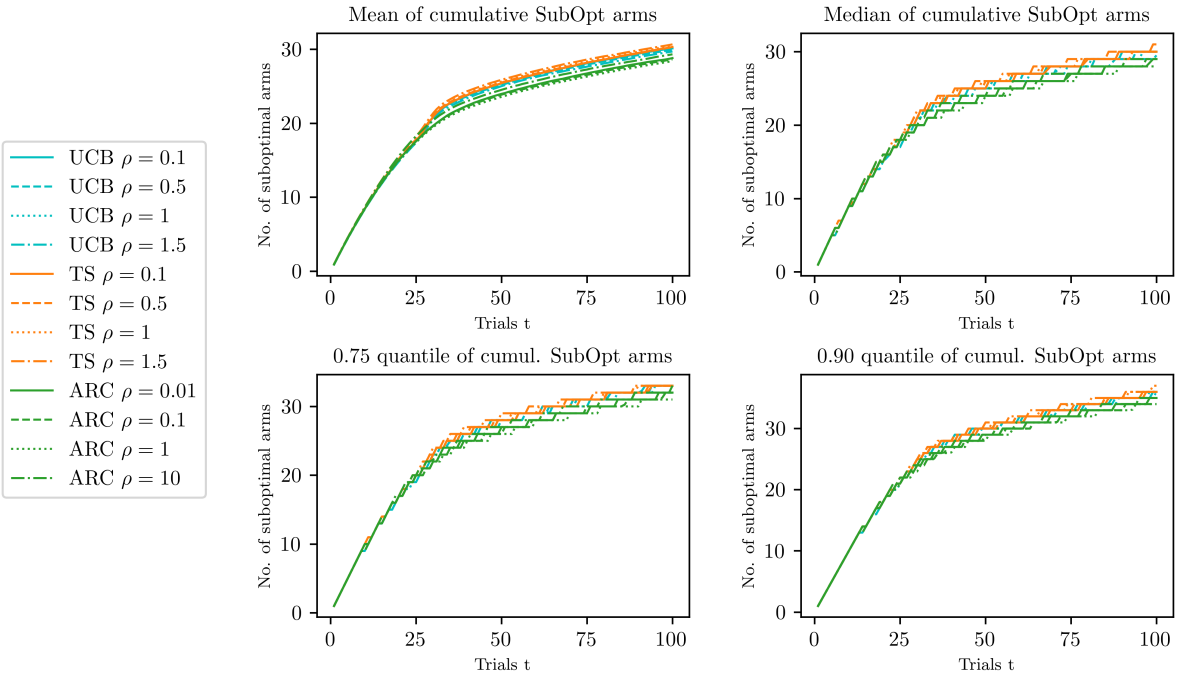

Figure 7: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Stochastic Linear Per Arm Environment

We again see that the performance of the algorithms is similar. In this particular example, ARC actually manages to slightly outperform Linear UCB and Linear Thompson Sampling, even though there is little additional structure for the ARC algorithm to exploit.

4.3 Per Arm Environment with Changing Variance

We now consider Per Arm Environment with Changing Variance environment, which provides additional structure that the ARC algorithm can exploit.

We first consider an extreme example. We run 1000 simulations of a 10-armed bandit with the dimension of the global context set to $l = 0$ and the dimension of the per-arm context set to $k = 3$, so the hidden parameter θ and the contexts $b_{i,t}$ lie in $\mathbb{R}^3$. Each entry in the hidden parameter θ is uniformly sampled in [0, 1], and we generate a new θ for each simulation. . At each time step, each per-arm context is sampled from the following distribution.

$$b_{i,t} = \begin{cases} [1,0,0] & \text{with probability } \frac{1}{10}, \\ [0,1,0] & \text{with probability } \frac{1}{10}, \\ [0,0,1] & \text{with probability } \frac{1}{10}, \\ [1,u,0.5-u] & \text{with probability } \frac{7}{30}, \\ [u,1,0.5-u] & \text{with probability } \frac{7}{30}, \\ [u,0.5-u,1] & \text{with probability } \frac{7}{30}, \end{cases}$$

where u is sampled uniformly in $[0, 0.5]$. In this extreme example, we set the variance function to be

$$\sigma^2_{i,t} = \begin{cases} 0.01 & \text{for } b_{i,t} \text{ of form [1,0,0], [0,1,0] or [0,0,1],} \\ 1 & \text{for } b_{i,t} \text{ of form } [1,u,0.5-u], [u,1,0.5-u] \text{ or } [u,0.5-u,1]. \end{cases}$$

In this construction, any actions with a context of the first type are very informative due to the low variance. However, as all elements in θ are positive, by construction such an action will likely not be optimal because the corresponding action of the second type (if available) will have a larger dot product with θ (as all terms are positive). Nevertheless, it is worth playing an action with each of the

possible contexts [1,0,0], [0,1,0] and [0,0,1] once towards the start of the game so that we obtain very good estimates of the hidden parameter, ensuring that later choices are very well informed.

We take the horizon to be $T = 100$ and therefore set $\beta = 1 - \frac{1}{T} = 0.99$ for the ARC algorithm accordingly. We set the exploration parameters ρ to be 0.1, 0.5, 1 and 1.5 for both Linear UCB and Linear Thompson Sampling, and ρ to be 0.01, 0.1, 1 and 10 for ARC. We obtain the following expected regret and suboptimal arm plots.

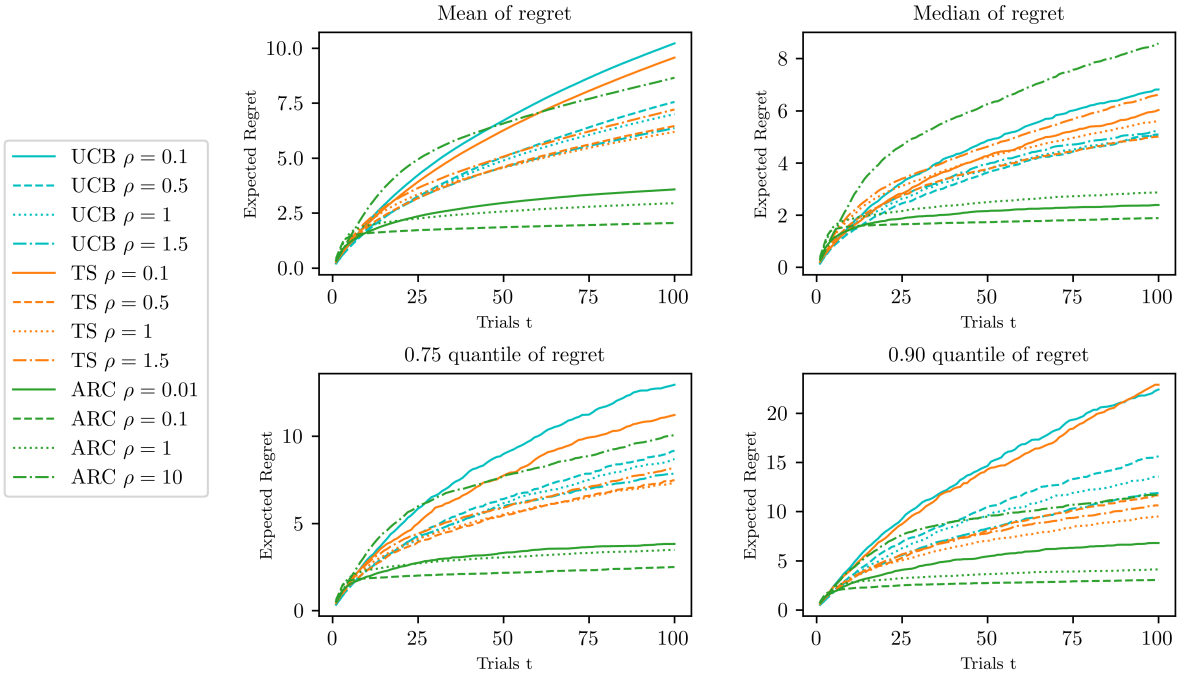

Figure 8: Plot of the cumulative regret for the different bandit algorithms in the Per Arm Environment with Changing Variance (extreme case)

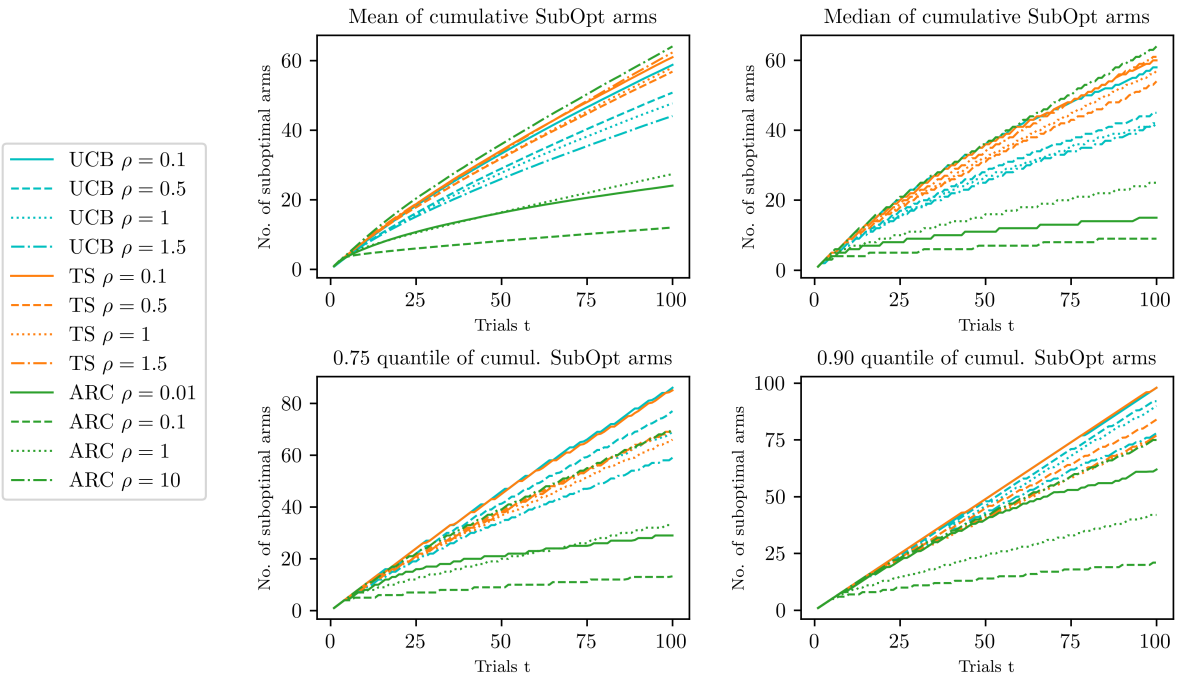

Figure 9: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Per Arm Environment with Changing Variance (extreme case)

We see that the ARC algorithm outperforms the Linear UCB and Linear Thompson Sampling algorithms in this example. For the ARC algorithm, taking $\rho = 0.1$ gives the best results, and taking ρ equal to 0.01 or 1 also gives very good performance. Taking $\rho = 10$ results in too much exploration. From the suboptimal arms plot (Figure 9) we see that in this example, the ARC algorithm can quickly identify the optimal action. This is because in the first few actions, it prioritises choosing the informative actions of the first type and thus obtains a very accurate estimate for the hidden parameter θ early on.

Indeed, to see this behaviour clearly, in the table below we show the proportions of choices that are of type [1,0,0], [0,1,0] or [0,0,1], as the game progresses. We show only for the best performing algorithm of each type, that is, with $\rho = 1.5$ for UCB, $\rho = 1$ for Thompson Sampling, and $\rho = 0.1$ for ARC.

Table 1: Proportion of actions chosen of type [1,0,0], [0,1,0] or [0,0,1].

	Time step(s) t							
Algorithm	1	2	3	4	5	6-10	11-20	21-100
UCB 1.5	0.0%	8.1%	14.3%	16.6%	15.4%	10.4%	5.21%	2.52%
TS 1	29.3%	28.2%	22.3%	22.9%	22.2%	17.9%	11.6%	5.26%
ARC 0.1	91.2%	79.4%	65.1%	34.2%	20.3%	9.20%	5.23%	3.46%

This behaviour allows the ARC algorithm to frequently identify the optimum action very early, leading to very strong performance. The other algorithms do not consider the learning value of the actions, so will likely not choose the more informative but costly actions and hence cannot exploit the additional structure as successfully.

Although this is quite an extreme example, it demonstrates the ability of the ARC algorithm to exploit additional structure in a bandit problem. We now consider another example where the variance of the reward is a continuous function of the context.

Again we consider the per-arm case with 10 actions with the dimension of the global context set to $l = 0$ and the dimension of the per-arm context set to $k = 3$, so the hidden parameter θ and the contexts $b_{i,t}$ lie in $\mathbb{R}^3$. Each entry in the hidden parameter θ is uniformly sampled in [0, 1] and a new θ

is generated for each simulation. At each time step we sample new per-arm contexts $b_{i,t}$ whose entries are uniformly sampled in [0, 0.5]. The variance in the reward for each action i at time t is given by the formula

$$\sigma_{i,t}^2 = \|b_{i,t}\|_{\ell_1}^4. \tag{23}$$

In this construction, an action whose context has a small sum is very informative but will likely have a lower expected reward (as all entries in the hidden parameter are positive) while an action whose context has a large sum is much less informative but will in general have a higher expected reward.

We run 100 simulations with the horizon set to $T = 1000$, and hence set $\beta = 1 - \frac{1}{T} = 0.999$ for the ARC algorithm. We set the exploration parameters ρ to be 0.5, 1, 1.5 and 2 for Linear UCB, and 0.05, 0.1, 0.5 and 1 for Linear Thompson Sampling. For ARC we set ρ to be 0.001, 0.01, 0.1 and 1. We obtain the following plots.

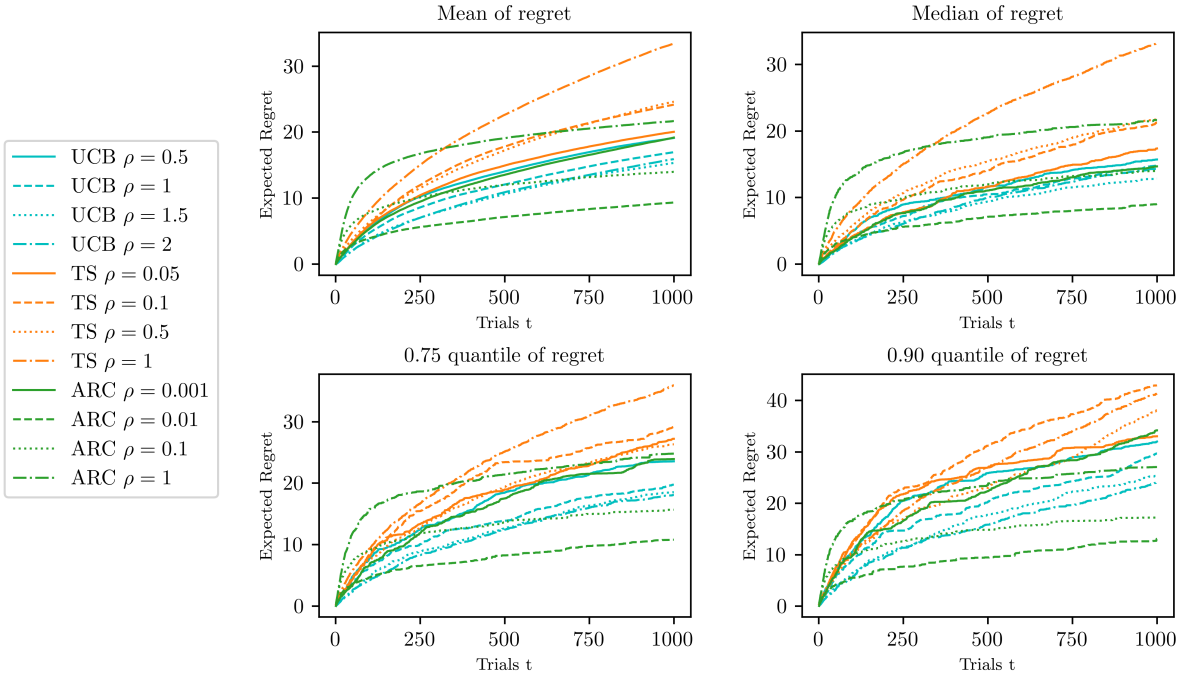

Figure 10: Plot of the cumulative regret for the different bandit algorithms in the Per Arm Environment with Changing Variance (with $\sigma_{i,t}^2 = \|b_{i,t}\|_{\ell_1}^4$)

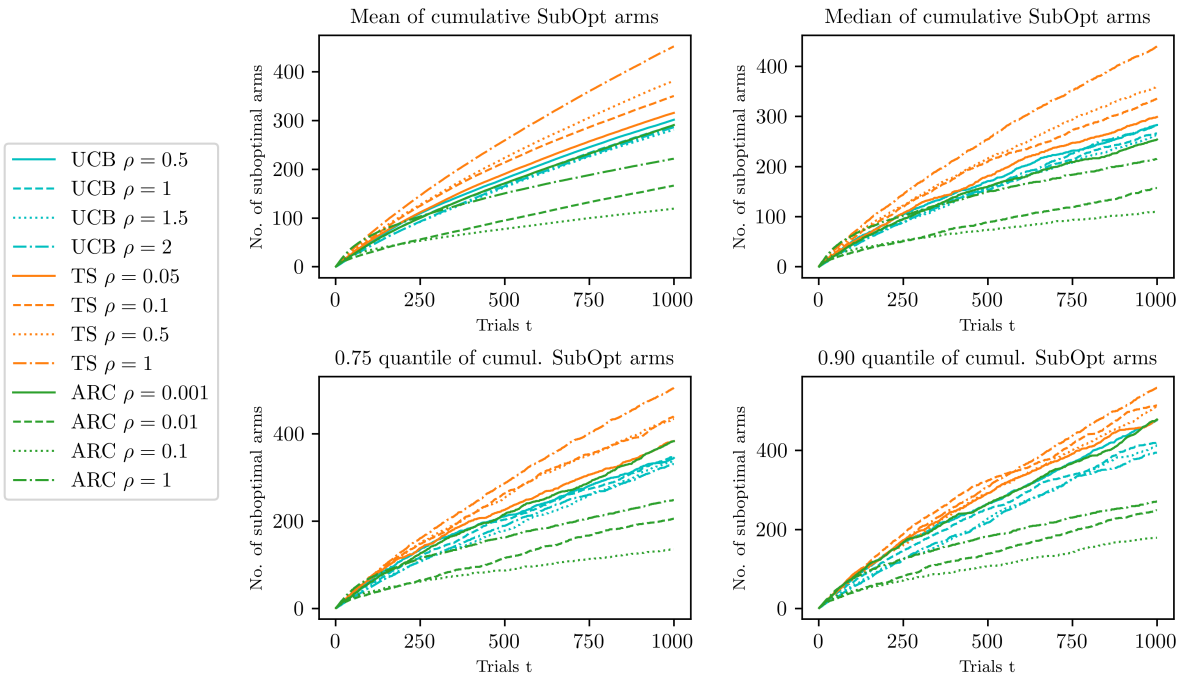

Figure 11: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Per Arm Environment with Changing Variance (with $\sigma^2_{i,t} = \|b_{i,t}\|^4_{\ell_1}$)

We see that ARC with $\rho = 0.01$ gives the best results, although the difference in the regret plots (Figure 10) is less pronounced than in the previous example. From the suboptimal arms plot (Figure 11) we see that the ARC algorithm is significantly better at choosing the optimum action than the alternatives, with $\rho = 0.1$ selecting the best action most frequently, followed by $\rho = 0.01$. The other algorithms again cannot exploit the additional structure. They will learn a good strategy, however it will likely not be as frequently optimal as the ARC algorithm as they do not have as good an estimate for the hidden parameter θ. The regret plot reflects this – the performance advantage of the ARC algorithm with $\rho = 0.01$ is less pronounced than in the suboptimal arms plot but is still clear. Several of the Linear UCB algorithms also achieve quite low regret (with $\rho = 1.5$ having the lowest mean regret) but do not as frequently identify the optimum action, which can be seen by the steeper gradient on the regret plot.

4.4 Graphical Bandit

We now consider a bandit construction that is a particular instance of the above per-arm bandit with changing variance. Consider a graph G with l vertices labelled 1 to l. Each vertex v has an associated hidden scalar θ_v which form the elements of the hidden parameter θ.

We use the same set-up as in the per-arm case above, and consider a 10-armed bandit where at each time step the context for each action is generated according to the following:

- Pick a central vertex, with the probability of picking each vertex proportional to the degree of that vertex.
- For each neighbour of this central vertex, randomly decide whether to include the neighbour (with the probability of inclusion being 0.4 each time).
- The per arm context is a vector with value 1 corresponding to the centre vertex and its included neighbours, and value 0 for the vertices not included.

This results in the reward of the chosen action having a mean given by the sum of the hidden parameters corresponding to the vertices included in the chosen action's context. To visualise this, we give a demonstration of the choice we must make at a given time step. Suppose we have the following graph

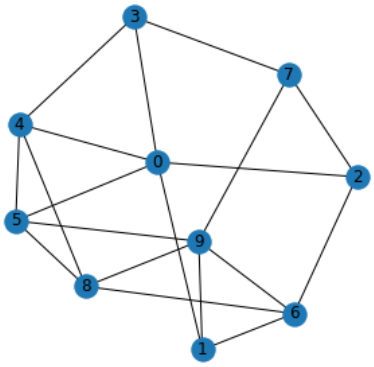

Figure 12: A graph

and that at the current time step we have sampled the per-arm contexts $b_{1,t}, \ldots, b_{10,t}$ which we suppose correspond to the following subgraphs highlighted in red.

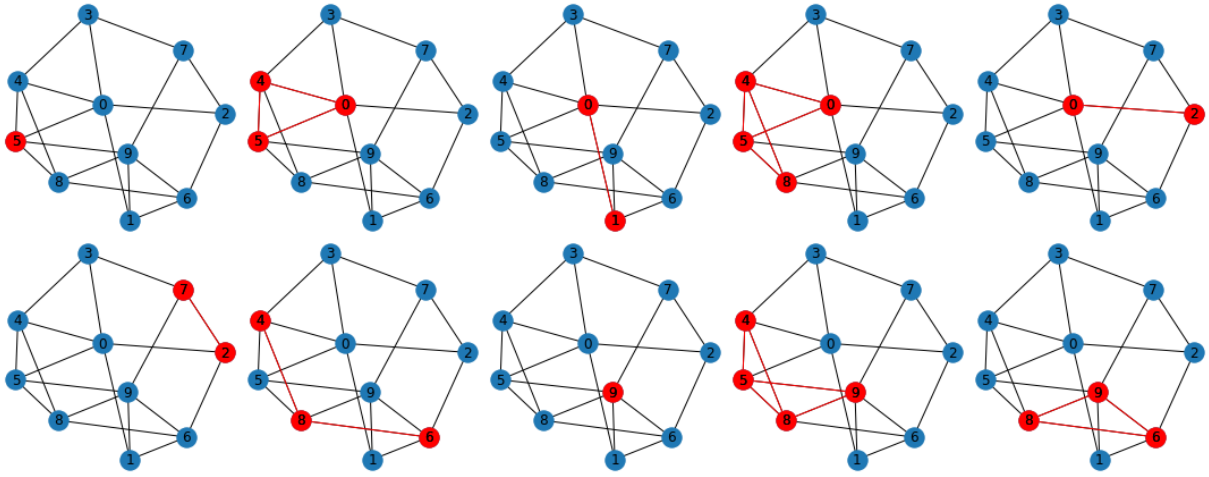

Figure 13: Subgraphs corresponding to the per arm contexts

To maximise the expected reward at this time step, we must choose the subgraph whose nodes have the largest sum of their hidden scalar parameters θ_v.

In our examples below, we consider working with graphs with 10 vertices. For each simulation we randomly generate a graph with order 10 (generated binomially using $p = 0.4$), so the hidden parameter θ and the contexts $b_{i,t}$ lie in $\mathbb{R}^{10}$. We generate a new hidden parameter θ for each simulation, and each entry θ_v is uniformly sampled in [-1, 1]. We sample the action contexts $b_{i,t}$ as described above. We consider two examples with different choices of variance function σ^2. In both examples we run 100 simulations with the horizon set to $T = 1000$, and therefore set $\beta = 1 - \frac{1}{T} = 0.999$ for the ARC algorithm.

In the first example we again consider an extreme case to demonstrate the advantages of the ARC algorithm. We choose the variance function to be

$$\sigma^2_{i,t} = \begin{cases} 0.001 & \text{for } \|b_{i,t}\|_{\ell_1} = 1, \\ 0.01 & \text{for } \|b_{i,t}\|_{\ell_1} = 2, \\ 0.5 & \text{for } \|b_{i,t}\|_{\ell_1} = 3, \\ 1 & \text{for } \|b_{i,t}\|_{\ell_1} \geq 4. \end{cases}$$

We set the exploration parameters ρ to be 0.1, 0.5, 1 and 1.5 for both Linear UCB and Linear Thompson Sampling, and ρ to be 0.0001, 0.001, 0.01 and 0.1 for ARC. We obtain the following plots.

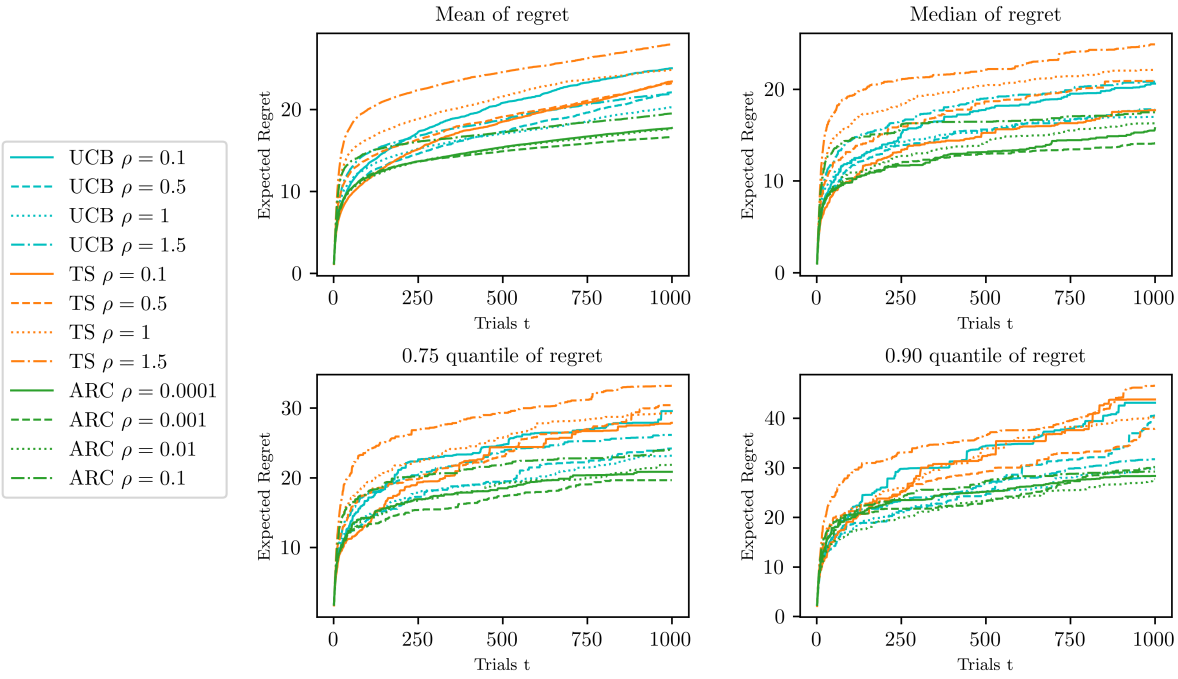

Figure 14: Plot of the cumulative regret for the different bandit algorithms in the Graphical Bandit (extreme case)

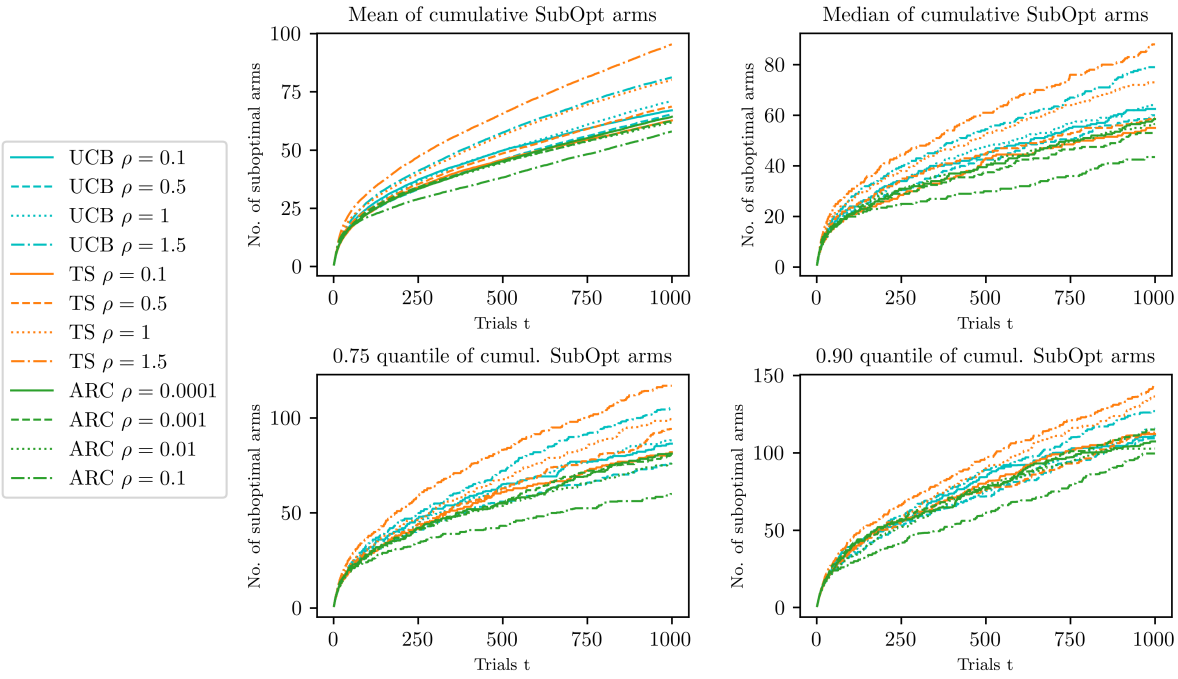

Figure 15: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Graphical Bandit (extreme case)

We see that the ARC algorithm outperforms the other algorithms by taking advantage of the addi-

tional information, choosing to gain information early on in order to exploit it later in the game. Taking $\rho = 0.001$ provides best performance on average, although taking ρ equal to 0.0001 or 0.01 also perform well. Taking $\rho = 0.1$ actually chooses the optimum action most often due to the increased exploration, but in doing so incurs a larger regret towards the start of the game leading to weaker performance overall.

In the second example, we instead use the following variance function:

$$\sigma_{i,t}^2 = \frac{\|b_{i,t}\|_{\ell_1}}{10}. \tag{24}$$

We obtain the following plots.

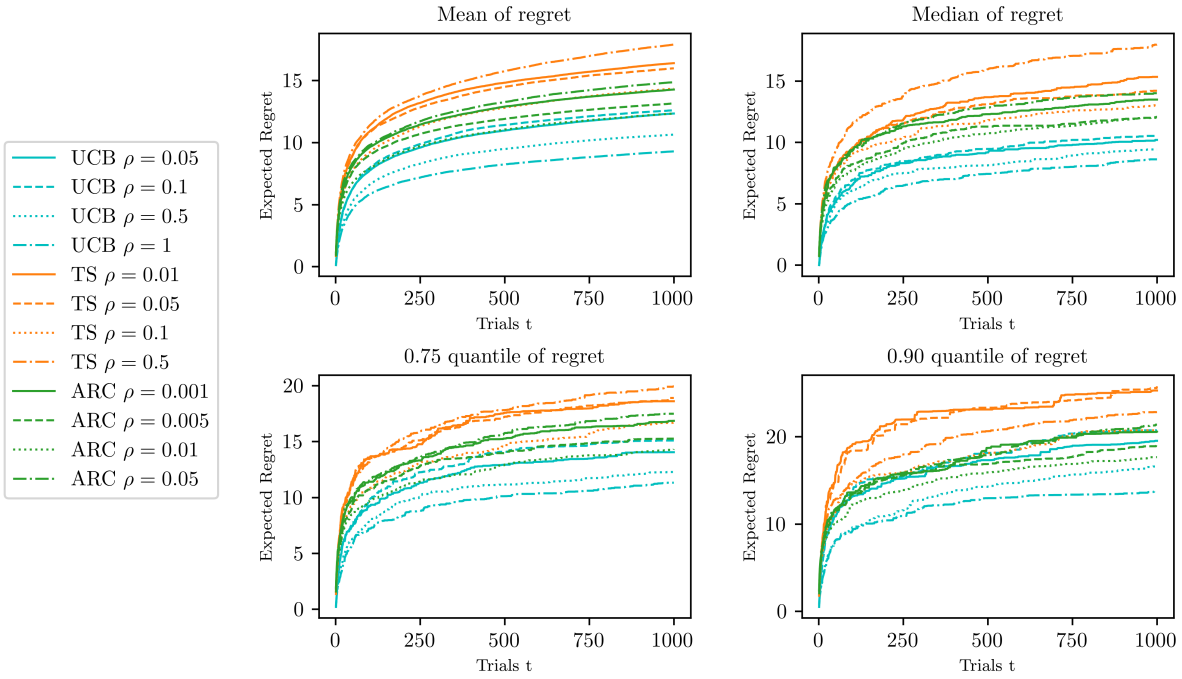

Figure 16: Plot of the cumulative regret for the different bandit algorithms in the Graphical Bandit (with $\sigma_{i,t}^2 = \frac{\|b_{i,t}\|_{\ell_1}}{10}$)

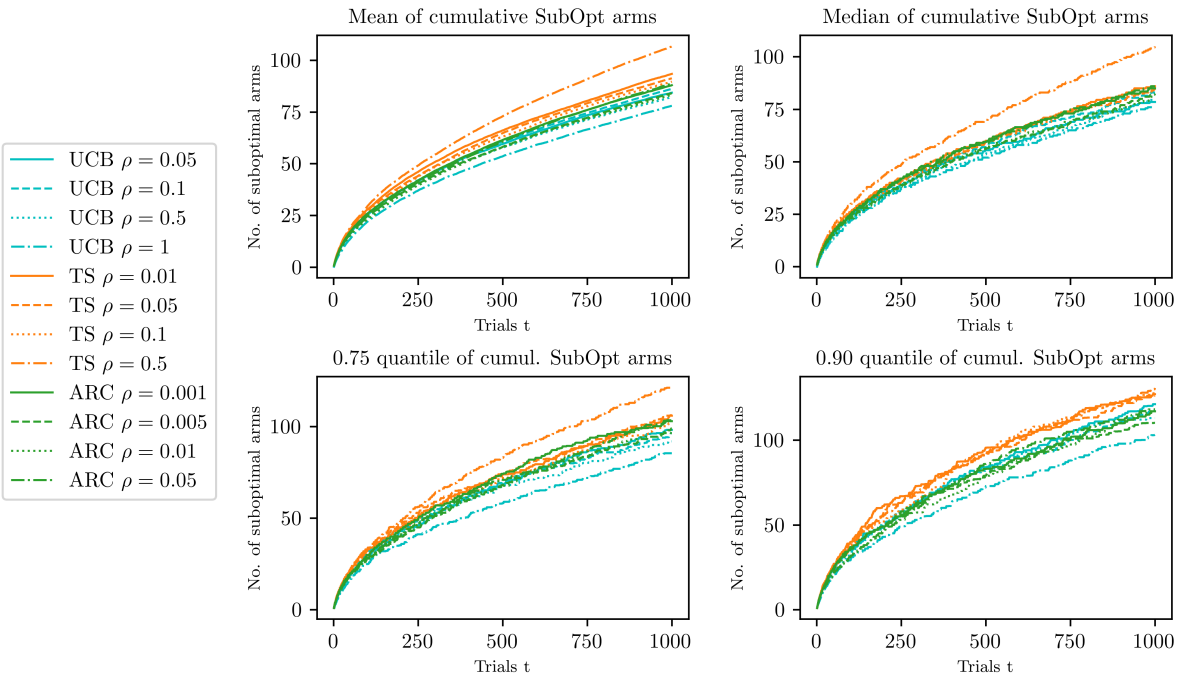

Figure 17: Plot of the cumulative no. of suboptimal arms chosen by the different bandit algorithms in the Graphical Bandit (with $\sigma^2_{i,t} = \frac{\|b_{i,t}\|_{\ell_1}}{10}$)

In this example we see that ARC performs better than Linear Thompson Sampling but slightly worse than Linear UCB, which performs very strongly in this case. It appears that although there is additional structure for ARC to exploit, the variance function does not provide large enough differences in the possible variances for ARC to have a significant advantage.

5 Conclusion

We adapted the ARC algorithm for contextual bandits and empirically assessed performance against the widely used Linear UCB and Linear Thompson Sampling algorithms in various examples using the TF-Agents library. We demonstrated that ARC can perform relatively well even in cases with little additional structure, but can often outperform the alternatives in our environment where the variance in the reward is a function of the chosen context. In particular, ARC performs well in cases where there are large differences in the possible variances in the reward. Further work can investigate in more detail which cases lend themselves to strong ARC performance, and also could adapt the proof of ARC algorithm in [3] to have rigorous foundations in the contextual bandits case. Investigation into the existence of the fixed point solution, and indeed the number of such solutions, would also be of interest, as would an extension to cases where the variance function is not known.

Acknowledgements

Samuel Cohen acknowledges the support of Alan Turing Institute under the Engineering and Physical Sciences Research Council grant EP/N510129/1. Samuel Howard acknowledges the EPSRC Vacation Internships scheme for funding this Summer Research Internship at the Oxford Mathematical Institute.

References

[1] Shipra Agrawal and Navin Goyal. “Thompson Sampling for Contextual Bandits with Linear Payoffs”. In: *Proceedings of the 30th International Conference on Machine Learn-*

ing. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 2013, pp. 127–135. URL: https://proceedings.mlr.press/v28/agrawal13.html.

[2] Alan Bain and Dan Crisan. "Fundamentals of Stochastic Filtering". In: 2008.

[3] Samuel N. Cohen and Tanut Treetanthiploet. *Asymptotic Randomised Control with applications to bandits*. 2020. arXiv: 2010.07252 [math.OC].

[4] William Cruz, José Martínez, and Marcos Raydan. "Spectral Residual Method without Gradient Information for Solving Large-Scale Nonlinear Systems of Equations". In: *Math. Comput.* 75 (July 2006), pp. 1429–1448. DOI: 10.1090/S0025-5718-06-01840-0.

[5] Sergio Guadarrama et al. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. https://github.com/tensorflow/agents. [Online; accessed 25-June-2019]. 2018. URL: https://github.com/tensorflow/agents.

[6] Rudolf E. Kálmán and Richard S. Bucy. "New Results in Linear Filtering and Prediction Theory". In: *Journal of Basic Engineering* 83 (1961), pp. 95–108.

[7] Johannes Kirschner and Andreas Krause. "Information Directed Sampling and Bandits with Heteroscedastic Noise". In: *Proc. International Conference on Learning Theory (COLT)*. 2018.

[8] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. DOI: 10.1017/9781108571401.

[9] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *CoRR* abs/1003.0146 (2010). arXiv: 1003.0146. URL: http://arxiv.org/abs/1003.0146.

[10] J. J. More, B. S. Garbow, and K. E. Hillstrom. "User guide for MINPACK-1. [In FORTRAN]". In: (Aug. 1980). DOI: 10.2172/6997568. URL: https://www.osti.gov/biblio/6997568.

[11] Daniel Russo. "A Note on the Equivalence of Upper Confidence Bounds and Gittins Indices for Patient Agents". In: *CoRR* abs/1904.04732 (2019). arXiv: 1904.04732. URL: http://arxiv.org/abs/1904.04732.

[12] Daniel Russo and Benjamin Van Roy. "Learning to Optimize Via Information Directed Sampling". In: *CoRR* abs/1403.5556 (2014). arXiv: 1403.5556. URL: http://arxiv.org/abs/1403.5556.

[13] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Calculation of L^λ

The calculation of the term L^λ used in the ARC algorithm is dependent on the choice of reward function. In our case the reward is linear, whereas the examples in [3] use a non-linearity in the reward function. Hence we must re-derive an expression for L^λ for our linear reward function in a similar manner to the derivation in [3, Appendix C.4].

Recall in our setup that if we choose action i, the reward $R_{i,t}$ is then sampled as

$$R_{i,t} \sim N(b_{i,t}^\top \theta, \sigma_{i,t}^2) \tag{25}$$

for some known variances $\sigma_{i,t}^2$.

Hence for each i we have

$$f_t^{(i)}(m, \Sigma, B_t) := \mathbb{E}_{m,\Sigma,B_t}(R_{i,t}) = b_{i,t}^\top m. \tag{26}$$

Taking derivatives we have

$$\partial_{m_j} f^{(i)} = (b_{i,t})_j, \tag{27}$$

$$\partial_{m_k m_j} f^{(i)} = 0, \tag{28}$$

$$\partial_{\Sigma} f^{(i)} = 0. \tag{29}$$

Substituting these values in to the expression in [3, Appendix C.4] we obtain the required expression for L in our bandit setup, namely

$$L^{\lambda}_{j,t}(a, m, \Sigma, B_t) = \frac{1}{2\lambda}\left(\frac{1}{s_{jj} + P_j^{-1}}\right)\left(\sum_{i=1}^{K} \nu_i^{\lambda}(a) s_{ij}^2 - \left(\sum_{i=1}^{K} \nu_i^{\lambda}(a) s_{ij}\right)^2\right) \tag{30}$$

where $s_{ij} = s_{ij}(\Sigma, B_t)$ is defined as

$$s_{ij}(\Sigma, B_t) = b_{i,t}^{\top} \Sigma \; b_{j,t}. \tag{31}$$

It is worth noting that in the derivation of the ARC algorithm in [3], the assumption that the contexts remain the same throughout the duration of time that the bandit is played is used to derive the fixed point calculation. The algorithm takes this into account when deciding on the information value gained by playing an action, and based on the observed contexts it may appear that information about certain parameters is more valuable than others. This assumption that the contexts are fixed throughout now no longer applies in the contextual case, however the current contexts are often a good indication of the contexts that will appear throughout the game and we have seen that the algorithm still performs well in the contextual case.