Tabular and Deep Reinforcement Learning for Gittins Index

Anonymous submission

Abstract

In the realm of multi-armed bandit problems, the Gittins index policy is known to be optimal in maximizing the expected total discounted reward obtained from pulling the Markovian arms. In most realistic scenarios however, the Markovian state transition probabilities are unknown and therefore the Gittins indices cannot be computed. One can then resort to reinforcement learning (RL) algorithms that explore the state space to learn these indices while exploiting to maximize the reward collected. In this work, we propose tabular (QGI) and Deep RL (DGN) algorithms for learning the Gittins index that are based on the retirement formulation for the multi-armed bandit problem. When compared with existing RL algorithms that learn the Gittins index, our algorithms have a lower run time, require less storage space (small Q-table size in QGI and smaller replay buffer in DGN), and illustrate better empirical convergence to the Gittins index. This makes our algorithm well suited for problems with large state spaces and is a viable alternative to existing methods. As a key application, we demonstrate the use of our algorithms in minimizing the mean flowtime in a job scheduling problem when jobs are available in batches and have an unknown service time distribution.

1 Introduction

Markov decision processes (MDPs) are controlled stochastic processes where a decision maker is required to control the evolution of a Markov chain over its states space by suitably choosing actions that maximize the long-term payoffs. An interesting class of MDPs are the multi-armed bandits (MAB) where given K Markov chains (each Markov chain corresponds to a bandit arm), the decision maker is confronted with a K-tuple (state of each arm) and must choose to pull or activate exactly one arm and collect a corresponding reward. The arm that is pulled undergoes a state change, while the state of the other arms remain frozen. When viewed as an MDP, the goal is to find the optimal policy for pulling the arms, that maximises the cumulative expected discounted rewards. In a seminal result, Gittins and Jones in 1974 proposed a dynamic allocation index (now known as Gittins index) for each state of an arm and showed that the policy that pulls the arm with the highest index maximizes the cumulative expected discounted rewards collected (Gittins 1974). Subsequently, the MAB problem and its variants have been successfully applied in a variety of applications such as A/B testing, Ad placements, recommendation systems, dynamic pricing, resource allocation and job scheduling (Lattimore and Szepesvári 2020). Particularly for the job scheduling problem, the Gittins index policy has been shown to be optimal in minimizing the mean flow time for a fixed number of jobs. In fact, the Gittins index policy is known to coincide with several scheduling policies that were historically proved to be optimal under different problem settings (Pinedo 2012; Gittins, Glazebrook, and Weber 2011; Scully 2023). Also note that in a dynamic setting of an $M/G/1$ queue with job arrivals, the Gittins index policy is an optimal scheduling policy that minimizes the mean sojourn time (Aalto, Ayesta, and Righter 2009; Scully, Grosof, and Harchol-Balter 2020). More recently, the Gittins index policy was shown to minimize the mean slowdown in an $M/G/1$ queue (Aalto and Scully 2023). See (Gittins 1974, 1979; Tsitsiklis 1994; Frostig and Weiss 2016) for various equivalent proofs on the optimality of the Gittins index policy for the MAB problem.

In a related development, Whittle in 1988 formulated the restless multi-armed bandit problem (RMAB) where the state of passive arms are allowed to undergo Markovian transitions (Whittle 1988). While the Gittins index policy is no longer optimal in this setting, a similar index policy based on the Lagrangian relaxation approach was proposed, now popular as the Whittle index policy. The Whittle index policy has been observed to have near optimal performance in some problems and in fact it has been shown to be asymptotically optimal in the number of arms (Weber and Weiss 1990; Verloop 2016). Note that Whittle index coincides with Gittins index when only a single arm is pulled and passive arms do no undergo Markovian transitions.

It is important to observe that the computation of Gittins or Whittle index requires the knowledge of the state transition probabilities and the reward structure for the arms. In most applications of the Gittins index policy, such transition probabilities are typically not known. One therefore has to use RL algorithms to learn the underlying Gittins index for different states of the arms. Duff was the first to propose a Q-learning based algorithm to learn the Gittins index that uses a novel 'restart-in-state-i' interpretation for the problem (Duff 1995). As an application, it was shown that the algorithm could learn the optimal scheduling policy minimizing the mean flowtime for preemptive jobs appear-

ing in batches per episode and whose service time distributions were unknown. More recently, there have been several works that provide tabular and Deep RL algorithms for the restless multi-armed bandit case. Avrachenkov and Borkar were the first to propose a Q-learning based algorithm that converges to the Whittle index under the average cost criteria (Avrachenkov and Borkar 2022). Robledo et. al. proposed a tabular Q-learning approach called QWI (Robledo et al. 2022a) and its Deep RL counterpart QWINN (Robledo et al. 2022b) and prove their asymptotic convergence. Nakhleh et al. also develop a deep reinforcement learning algorithm, NeurWIN, for estimating the Whittle indices (Nakhleh, Hou et al. 2022). Note that when passive arms do not undergo state transitions, the preceding learning algorithms ('restart-in-state', QWI, QWINN, NeurWIN) also converge to the true Gittins index. See (Gast, Gaujal, and Yan 2023; Akbarzadeh and Mahajan 2023) for recent work on model based RL algorithms for restless bandits.

In this work, we propose a tabular algorithm called QGI (Q-learning for Gittins index) and Deep RL algorithm DGN (Deep Gittins Network) for learning the Gittins index of states of a multi arm bandit problem. Both these algorithms are based on the retirement formulation (Gittins, Glazebrook, and Weber 2011; Frostig and Weiss 2016). The DGN algorithm in particular leverages a DQN to learn the indices (Mnih et al. 2015). Compared to the existing algorithms, QGI and DGN have several common as well as distinguishing features. As in case of QWI, our algorithms also have a timescale separation for learning the Q-values and the indices. However, due to the nature of the retirement formulation, we are not required to learn the state-action Q-values for the passive action. We therefore need to learn a smaller Q-table as compared to QWI or the restart-in-state algorithm. Similarly in the DGN algorithm, we do not require experience replay tuples corresponding to the passive action (these are required in QWINN). Because of this, our algorithms have a significantly lower runtime and are more stable to changes in the hyperparameters. In fact, across our experiments, we found the convergence of QGI to be more robust to changes in the hyperparameters, as compared to QWI (see supplementary material for more details). The main contributions of this work are as follows:

- We propose a novel tabular and Deep Learning based approach to learn the Gittins index based on the retirement formulation.
- We show that the proposed algorithms have a superior performance over existing algorithms. More specifically, our algorithms offer lower runtime, better convergence and illustrate a lower empirical regret.
- We also prove asymptotic convergence of the Q values arising from the QGI algorithm (Theorem 1).
- We illustrate the application of our algorithms in learning the optimal scheduling policy that minimizes the mean flowtime for a fixed set of jobs with unknown service time distributions.

The rest of this paper is organized as follows: Section 2 provides preliminaries on the Gittins index. In Section 3 and Section 4, we propose the tabular QGI algorithm and the Deep RL based DGN algorithm respectively and compare its performance to state-of-art algorithms. In Section 5 we use the proposed algorithms to learn the optimal scheduling policy minimizing the flowtime when service time distributions are unknown.

2 Preliminaries on the Gittins index

Consider a multi-armed bandit problem with K (possibly heterogeneous) arms. Let $\mathcal{S}^i$ denote the state space for the i^{th} arm. We denote the random state and action for the i^{th} arm at the n^{th} time step by $s_n(i)$ and $a_n(i)$ respectively where $s_n(i) \in \mathcal{S}^i$ for all n. Note that $a_n(i) = 1$ if the i^{th} arm is pulled at time n and $a_n(i) = 0$ otherwise. Since exactly 1 arm is pulled each time, we have $\sum_i a_n(i) = 1$. Upon pulling the i^{th} arm, we observe a state transition to $s_{n+1}(i)$ with probability $p(s_{n+1}(i)|s_n(i), a_n(i))$ and receive a reward $r^i(s_n(i), a_n(i))$. Furthermore, when $a_n(i) = 0$, we assume that $r^i(s_n(i), 0) = 0$. The objective is to choose a policy π^* that maximizes the expected total discounted reward for a discount factor γ $(0 < \gamma < 1)$. The optimal policy is essentially a solution to the following optimization problem:

$$V_{\pi^*}(\bar{s}) := \max_{\pi} E\left[\sum_{t=0}^{\infty}\sum_{i=1}^{K}\gamma^t r^i(s_t(i), a_t(i))\right] \quad (1)$$

for any starting state $\bar{s} = (s_0(1), \ldots, s_0(K))$. In a seminal work, Gittins defined the bandit allocation index in (Gittins 1974) which is a solution technique for the above MAB problem. In fact, the Gittins index for arm i in state x, denoted by $G^i(x)$ is given by:

$$G^i(x) = \sup_{\sigma>0} G^i(x,\sigma) \text{ where}$$

$$G^i(x,\sigma) = \frac{\mathbf{E}\left\{\sum_{t=0}^{\sigma-1}\gamma^t r^i(s_t(i),1) \mid s_0(i) = x\right\}}{\mathbf{E}\left\{\sum_{t=0}^{\sigma-1}\gamma^t \mid s_0(i) = x\right\}}.$$

Here, $G^i(x,\sigma)$ is the expected discounted reward per expected unit of discounted time, when the arm is operated from initial state x, for a duration σ. The supremum here is over all positive stopping times σ. In fact, it turns out that this supremum is achieved by the stopping time $\tau(x) = \min\{t : G^i(s_t(i))) < G^i(x)\}$, which is the time where the value of Gittins index drops from $G^i(x)$ for the first time while continuously pulling arm i. See (Frostig and Weiss 2016) for more details.

3 QGI: Q learning for Gittins index

In this section, we present a Q-learning based algorithm for learning the Gittins index (QGI for short) that is based on the retirement formulation proposed by Whittle (Whittle 1980). See supplementary material for a review of state-of-art algorithms 'restart-in-state-i' (Duff 1995) and QWI (Robledo et al. 2022a).

The retirement formulation: First assume you have a single arm that is in state x. For this arm, you can either pull the arm and collect reward or choose to retire and receive a

terminal reward M. Let us denote the optimal value function in state x by $V_r(x, M)$. The actions are denoted by 1 (to continue) and 0 (to retire). The Bellman optimality equation for this problem is:

$$V_r(x, M) = max\{Q_M(x, 1), Q_M(x, 0)\} \text{ where}$$

$$\begin{aligned} Q_M(x,1) &= r(x,1) + \gamma \sum_j p(j \mid x, 1) \times \\ &\quad \max\{Q_M(j,1), Q_M(j,0)\} \text{ and} \\ Q_M(x,0) &= M. \end{aligned}$$

As shown in (Whittle 1980), this results in an optimal stopping problem (how long should you choose action 1 before retiring) and the Gittins index for state x is given by $G(x) = M(x)(1-\gamma)$ where

$$M(x) = inf\{M : V_r(x, M) = M\}. \tag{2}$$

Clearly, $G(x)$ can be obtained by finding the smallest value of M where $V_r(x, M) = M$. It is important to note here that $V_r(x, M)$ is bounded, convex and non-decreasing in M (Frostig and Weiss 2016; Whittle 1980).

The tabular QGI algorithm: Now suppose that state x is fixed as a reference state and the retirement amount is set to $M(x)$ which is proportional to the Gittins index for state x. In that case, the Bellman equations are

$$V_r(x, M(x)) = max\{Q_{M(x)}(x, 1), Q_{M(x)}(x, 0)\} \text{ where}$$

$$Q_{M(x)}(x,1) = r(x,1) + \gamma \sum_j p(j|x,1) max\{Q_{M(x)}(j,1), Q_{M(x)}(j,0)\}$$

$$Q_{M(x)}(j,1) = r(j,1) + \gamma \sum_k p(k|j,1) max\{Q_{M(x)}(k,1), Q_{M(x)}(k,0)\}$$

$$\text{and } Q_{M(x)}(j,0) = M(x) \text{ for all } j\ .$$

Now in a setting where the transitions probabilities are unknown, our goal is to learn the Gittins index for each state in all arms. For simplicity let us assume homogeneous arms and consider one such arm. For this arm, our aim is to learn the indices $M(x)$ for every $x \in \mathcal{S}$. Note that in the Bellman equations above, x plays the role of a reference state and to come up with appropriate Q-learning equations, we need an additional dimension in the Q-table for the reference state. Also note that since $M(x) = Q_{M(x)}(j, 0)$ for all j in the Bellman equations above, we can have an iterative update equation for $M(x)$ and completely ignore the entries corresponding to $Q_{M(x)}(j, 0)$ from the Q-table.

For a K-armed bandit in state $(s_n(0), s_n(1), ...s_n(K))$ in the n^{th} step, the agent selects an arm via an ϵ-greedy policy based on the current estimates of indices, denoted by $(M_n^0(s_n(0)), \ldots, M_n^K(s_n(K))$. Upon pulling an arm, say i, the agent observes a transition from state $s_n(i)$ to $s_{n+1}(i)$, and a reward $r^i\left(s_n(i)\right)$. As earlier, since arms are homogeneous, we will denote $s_n(i)$ by s_n, $M_n^i(s_n(i))$ by $M_n(s_n)$ and $r^i\left(s_n(i)\right)$ by $r(s_n)$. This reward $r(s_n)$ can be used to update the Q-values for s_n for each reference state leading to N Q-learning updates. This is followed with pushing current estimate of $M(\cdot)$ in direction of $V_r(\cdot, M)$, motivated by equation (2). This gives rise to the following pair of update steps in the chosen arm upon observing each transition. For each reference state $x \in \mathcal{S}$, when an arm in state s_n is pulled, we perform:

$$\begin{aligned} &Q_{n+1}^x\left(s_n, 1\right) = (1-\alpha(n)) Q_n^x\left(s_n, 1\right) + \\ &\alpha(n)\left(r\left(s_n\right) + \gamma \max\left\{Q_n^x(s_{n+1}, 1), M_n(x)\right\}\right) \end{aligned} \tag{3}$$

and

$$\begin{aligned} M_{n+1}(x) &= (1-\beta(n)) M_n(x) \\ &+ \beta(n)\left(\max\left\{Q_{n+1}^x(x,1), M_n(x)\right\}\right). \end{aligned} \tag{4}$$

For ease of exposition, we have assumed here that all arms are homogeneous and therefore in the above equations, the same Q-table entries are updated for different arms visiting the same state. It is also important to note that we have not used $Q_n^x(s_{n+1}, 0)$ to bootstrap and so we need not update those values. In fact, we do not even need to store them. For a guaranteed convergence to the true Gittins indices (see Theorem 1), the learning rate sequence $\alpha(n)$ and $\beta(n)$ are chosen to satisfy $\sum_n \alpha(n) = \infty, \sum_n \alpha(n)^2 < \infty$, $\sum_n \beta(n) = \infty, \sum_n \beta(n)^2 < \infty$. We also require that $\beta(n) = o(\alpha(n))$ that allows for two distinct time scales, namely, a relatively faster time scale for the updates of the state-action function, and a slow one for the Gittins indices. At this point, it would be a good idea to compare and contrast our learning equations with that of QWI (Equations (5),(6)) and restart-in-state algorithm (Equation (7)) which are recalled here for convenience. The details for both these update rules can be found in the supplementary material.

$$\begin{aligned} &Q_{n+1}^x(s_n, a_n) = (1-\alpha(n)) Q_n^x(s_n, a_n) + \alpha(n) \times \\ &((1-a_n)\lambda_n(x) + a_n r(s_n) + \gamma \max_{v \in \{0,1\}} Q_n^x(s_{n+1}, v)) \end{aligned} \tag{5}$$

$$\lambda_{n+1}(x) = \lambda_n(x) + \beta(n)\left(Q_n^x(x,1) - Q_n^x(x,0)\right) \tag{6}$$

$$\begin{aligned} Q^k(s_n, 1) &= (1-\alpha(n)) Q^k(s_n, 1) \\ &+ \alpha(n)\left[r(s_n) + \gamma \max_{a \in \{0,1\}} Q^k(s_{n+1}, a)\right] \end{aligned} \tag{7}$$

It is quite evident that the update equations (5), (7) are performed for both actions $a_n = 1$ and $a_n = 0$ and in fact $Q_n^x(x, 0)$ is also directly involved in the update of the Gittins indices (see equation (6)). In contrast, there is no Q-value term corresponding to retirement action in both equations (3) and (4). We only consider $a_n = 1$ for the Q-table, reducing its size by a factor of 2. Let us now carefully observe Eq. (4). We see that $V_{r,n}(x, M_n(x)) = \max\left(Q_n^x(x,1), M_n(x)\right)$ is being invoked to push $M_n(x)$ in its direction. Let us recall that $V_r(x, M)$ is bounded, non-increasing and convex w.r.t M. Hence, one of two following cases arise. Either, M is large and hence it dominates, making $V_r(x, M) - M = 0$, in which case (4) is not updated. It

Algorithm 1: QGI for N states, K arm bandits

Require: Discount parameter $\gamma \in (0,1)$, exploration parameter $\epsilon \in [0,1]$

1: Initialise M (NxK) and Q (NxNxK) matrix for all states in each arm
2: Initialize s_0 for all arms
3: **for** $n = 1$ **to** n_{end} **do**
4: Select an arm i to pull through ϵ-greedy policy
5: Get new state $s_{n+1}(i)$ and reward $r(s_n(i))$ from state $s_n(i)$
6: Update learning rate $\alpha(n)$, $\beta(n)$
7: Update $Q_n^x(s_n(i),1)$ for all $x \in \mathcal{S}$ via (3)
8: Update $M_n(x)$ for all $x \in \mathcal{S}$ via (8)
9: **end for**

is only if $Q_n^x(x,1) > M_n(x)$, $M_n(x)$ is updated in Eq. (4). Therefore, in Eq. (4), $M_n(x)$ needs to initialised by a value which is known to be smaller than the Gittins index in that state. To avoid imposing this restriction, we simply replace $V_{r,n}(x,M)$ by $Q_n^x(x,1)$, and this allows us to initialise M to arbitrary value. We therefore use the following equation in QGI in place of Eq. (4):

$$M_{n+1}(x) = M_n(x) + \beta(n)\left(Q_{n+1}^x(x,1) - M_n(x)\right) \quad (8)$$

We are now in a position to present the QGI algorithm (Algorithm 1) that essentially makes use of Eq. (3) and (8). Let $N := |\mathcal{S}|$ denote the number of states of an arm. For a fixed ϵ and γ, we arbitrarily initialise the initial states of the K arms. We then initialize a Q matrix of size $N \times N \times K$ and a vector M of size $N \times K$. If the arms are homogeneous, then the Q table has a size of $N \times N$. We then choose the best arm (arm with the highest estimate of Gittins index) with a probability of $1-\epsilon$, and choose an arm randomly with probability ϵ. In the chosen arm i, we observe a state transition from s_n to s_{n+1} with a reward $r(s_n)$. We use this to update our Q-values and M vector as in Eq. (3) and (8). Owing to the nature of updates presented in Eq. (3) and (8), while the time complexity in QWI and the restart-in formulation per iteration is the same, we have the following two-fold advantage leading to lower runtime and smoother convergence (see numerical results):

- The effective updates in each iteration to the Gittins index values are N, compared to just 1 in restart-in-state. We have observed that this leads to better convergence of the Gittins index values (see supplementary material for extensive results). For homogenuous case, NK updates are done to Q values and N updates to λ in each iteration for QWI when $\beta \neq 0$. Out of those NK updates, $N(K-1)$ updates are done to $Q^x(s_n,0)$ for all arm and for all x and only N updates to the values corresponding to $a_n = 1$. In contrast, QGI does N updates only to $Q^x(s_n,1)$ for all x as Q-values for $a_n = 0$ is neither stored nor updated.
- Since, $Q_n^x(s_{n+1},0) = M_n(x)$ for all s_n, we do not track and update values of $Q_n^x(s_n,0)$ for every x and s_n resulting in significant space saving. In restart-in-state and QWI, the size of Q-table for each arm is $N \times N \times 2$. Here the first N corresponds to all reference states, the second N corresponds to all states and 2 for the actions of continue or stop. On the other hand, in QGI the Q-table for each arm is of the size $N \times N$.

The space saving discussed above is significant when the number of states or arms are very large. For example, for a 10 armed bandit with 100 states, we can observe that the Q-table will have $2,00,000$ entries for the restart-in-state algorithm, 2,01,000 entries for QWI while only 1,01,000 entries in QGI. We find this as a very useful feature, particularly for learning the optimal scheduling policy minimizing mean flowtime for a fixed number of jobs having arbitrary but continuous service time distributions. See supplementary material for such an example.

We state the following theorem that guarantees convergence of the Gittins index using the QGI algorithm. Proof is given in supplementary material for completeness.

Theorem 1. *Given learning rate sequences $\alpha(n)$ and $\beta(n)$ such that $\sum_n \alpha(n) = \infty, \sum_n \alpha(n)^2 < \infty$, $\sum_n \beta(n) = \infty, \sum_n \beta(n)^2 < \infty$ and $\beta(n) = o(\alpha(n))$, iterative equations (3) and (8) converge to the optimal state-action values for the Gittins index policy $Q_G(s,a)$, and to the Gittins indices $G(s) = (1-\gamma)M(s)$, where $M_n(s) \to M(s)$ and $Q_n(s,a) \to Q_G(s,a)$ for all $s \in S$, $a \in A$ as $n \to \infty$.*

4 DGN: Deep Gittins Network

The DGN (Deep Gittins Network) algorithm is based on learning the Gittins estimates through a neural network instead of the tabular method of QGI. Note that DGN is also based on the the retirement formulation and much of the discussion leading up to Equations (3) and (8) is also relevant to this Deep RL formulation. We use a neural network with parameters θ as a function approximator for the state action value function $Q^x(s,1)$. The input to the neural network is the current state s_n and reference state x and the first output cell represents $Q_\theta^x(s,1)$. As in QGI, we use a separate stochastic approximation equation to track the Gittins index $M(x)$ in state x. Our DGN algorithm utilises a DQN architecture with soft updates for better convergence guarantee (Mnih et al. 2015). Therefore a second (target) neural network with parameters θ' is used to calculate the target Q-values by bootstrapping with our current estimate. These are denoted by $Q_{\theta'}^x(s_{n+1},1)$. To further improve stability and reduce variance in target estimates, we do a soft update to θ' values instead of copying the θ values directly to the target network (Lillicrap et al. 2016). The soft updating for θ' is done using the iteration $\theta' = \tau\theta' + (1-\tau)\theta$, once in every κ number of steps.

As in QGI, the agent starts by pulling an arm i in state $s_n(i)$ via an ϵ-greedy policy based on the current estimate of Gittins indices. This results in a transition to state $s_{n+1}(i)$ denoted by $s'_n(i)$ for notational convenience and a reward $r^i(s_n(i))$ is observed.The observed state transitions form an experience tuple that is denoted by $(s_n(i), a_n(i), r^i(s_n(i)), s'_n(i))$. These tuples are stored along with other tuples in what is called as a replay buffer. In fact, we utilise this one tuple from memory to efficiently create N different tuples while sampling minibatches by

considering every reference state $x \in \mathcal{S}$. Each state transition is therefore used to update the Q-values for all reference states x, and to generate N tuples as mentioned above. Note that since we are in the Gittins setting, there are no rewards from passive action and therefore we do not create experience tuples corresponding to action $a_n = 0$. This results in a replay buffer with a smaller size as compared with the replay buffer of QWINN where experience tuples corresponding to action 0 are stored as well. After collecting enough experience tuples, we randomly create a minibatch of size B to iterate upon. For each experience tuple $\mathcal{K} = ([s_k(i), x], a_k(i), r^i(s_k(i)), [s'_k(i), x])$ in a minibatch, we first pass the pair $(s_k(i), x)$ through the primary network to get $Q^x_\theta(s_k(i), 1)$ for all $x \epsilon \mathcal{S}$. Then the pair $(s'_k(i), x)$ is fed to the target network to get $Q^x_{\theta'}(s'_k(i), 1)$ values. Lastly, we consider the most recent value of $M_n(x)$ available, allowing us to compute $Q^x_{\text{target}}(s_k(i), 1)$ values as following:

$$\begin{aligned} Q^x_{\text{target}}(s_k(i), 1) &= r(s_k(i)) + \\ &\quad \gamma \max\left(Q^x_{\theta'}(s'_k(i), 1), M_n(x)\right) \end{aligned} \tag{9}$$

Once $Q^x_{\text{target}}(\cdot)$ and $Q^x_\theta(\cdot)$ are calculated for the minibatch, we calculate the mean-squared loss as follows:

$$\text{MSE Loss} = \frac{1}{B}\sum_{k=1}^{B}\sum_{x=1}^{N}(Q^x_{\text{target}}(s_k(i), 1) - Q^x_\theta(s_k(i), 1))^2 \tag{10}$$

where $s_k(i)$ is the state of the k^{th} sample in the replay buffer. Then we update the θ values in the neural network via an appropriate optimizer (we use the Adam optimiser for our numerical results). In our implementation, we collect experience tuples and add them to the replay buffer in every step but do the parameter update steps along with the soft update once in every $\kappa = 10$ steps. This is coupled with the stochastic approximation update for M as follows:

$$M_{n+1}(x) = M_n(x) + \beta(n)\left(Q^x_\theta(x, 1) - M_n(x)\right) \tag{11}$$

We repeat the process for each sampled minibatch until convergence of the Gittins estimates. The neural network in our experiment consists of three hidden layers with (64,128,64) neurons. For the activation function, ReLU is used. This algorithm is formulated as shown in Algorithm 2.

Algorithm 2: DGN Algorithm

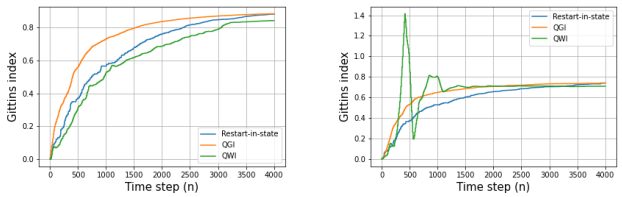
Require: Minibatch size B, Soft update parameter $\tau \in [0, 1]$, $\kappa \in \mathcal{Z}$, $\gamma \in (0, 1)$, $\epsilon \in [0, 1]$ and Learning rate β.
Return: Gittins index vector
for $n = 1$ **to** n_{terminal} **do**
  Update learning rate $\beta(n)$
  Pull an arm i through ϵ-greedy policy
  Do a transition from state $s_n(i)$ to $s_{n+1}(i)$ and observe reward $r(s_n(i))$
  Store the experience replay tuple into replay buffer
  if Size of memory > B & (n modulo $\kappa == 0$) **then**
    Sample a minibatch $\mathcal{B}$ of size B
    for $\mathcal{K} \in \mathcal{B}$ **do**
      for $x \in S$ **do**
        Predict $Q^x_\theta(s_k(i), 1)$ values
        Compute target $Q^x_{\text{target}}(s_k(i), 1)$ as in Eq. (9)
      end for
    end for
    Compute MSE loss between Q_θ and Q_{target} as in Eq. (10)
    Update the θ parameters through backpropagation
    Update target parameters as $\theta' = \tau\theta' + (1 - \tau)\theta$
  end if
  Update M (Retirement reward) via (11)
end for

Elementary example

We now illustrate the efficacy of the QGI and DGN algorithm on a slightly modified version of the restart problem from (Robledo et al. 2022b). The problem was modified to work for rested bandits (passive arms do not undergo state transitions). Here, the state space $S = \{0, 1, 2, 3, 4\}$ and there are 5 homogeneous arms. The transition probability matrix in the case of active action is:

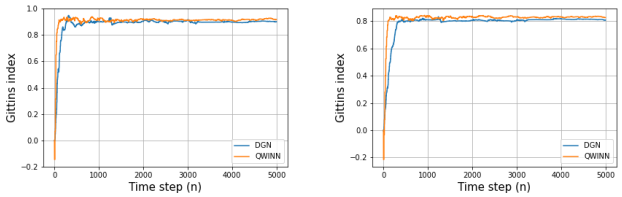
$$P(s'|s, 1) = \begin{bmatrix} 0.1 & 0.9 & 0 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 & 0 \\ 0.1 & 0 & 0 & 0.9 & 0 \\ 0.1 & 0 & 0 & 0 & 0.9 \\ 0.1 & 0 & 0 & 0 & 0.9 \end{bmatrix}$$

The reward for pulling an arm in state s is given by $r(s) = 0.9^s + 1$. The Gittins index can be analytically calculated in this problem, yielding G(0) = 0.9, G(1) = 0.819, G(2) = 0.74804, G(3) = 0.6909, G(4) = 0.64565 using a discount factor γ = 0.9. For the tabular experiments, we choose $\epsilon = 0.2$. While trying out different learning rate combinations that work for the two time scale algorithms (QWI and QGI) for the aforementioned problem, we observed that QWI was very sensitive to the hyperparameters of the learning rates chosen. Considering this, we decided to choose the structure of learning rates provided in (Robledo et al. 2022a). We identified hyperparameters: x, y, θ, κ and ϕ based on: $\alpha(n) = \frac{x}{\lceil \frac{n}{\theta} \rceil}$ and $\beta(n) = \frac{y}{1+\lceil \frac{n \log n}{\kappa} \rceil}\mathbb{I}_{n \mod \phi \equiv 0}$. Through an extensive search over 4200 hyperparameter combinations, and a refining process we choose the following learning rates for QGI: $x = 0.2$, $y = 0.6$, $\theta = 5000$, $\kappa = 5000$ and $\phi = 10$ and for QWI: $x = 0.1$, $y = 0.2$, $\theta = 5000$, $\kappa = 5000$ and $\phi = 10$. To check the details of the tuning process and a convergence analysis of QWI and QGI, refer to supplementary material. To maintain uniformity across algorithms, we constructed an ϵ-greedy version (with ϵ = 0.2) for restart-in-state as opposed to the original algorithm which does action selection via Boltzmann temperature scheduling. We ran all three algorithms for this experiment and the respective indices are shown in Fig. 1. Clearly, the QGI algorithm converges quickly compared to the restart-in-state algorithm to the true index value and is also less noisy as compared with QWI. For the Neural Network based algorithms, we tuned the hyperparameters by performing a grid search as well. This gave us the following set of hyperparameters for DGN: $B = 32$, $\tau = 1e-3$, $\alpha = 5e-3$ $\kappa = 10$ and $\beta(n) = 1 \times \mathbb{I}_{n \mod 5 \equiv 0}$ and for QWINN: $B = 32$, $\tau =$



(a) State 0 (b) State 2

Figure 1: (a), (b) show the comparative analysis of tabular methods on 4.1.



(a) State 0 (b) State 1

Figure 2: (a), (b) show the comparative analysis of neural network methods on 4.1.

$1e-3$, $\alpha = 5e-3$ $\kappa = 10$ and $\beta(n) = 0.2 \times \mathbb{I}_{n \mod 10 \equiv 0}$. The value of ϵ was kept as 1 for the experiment as in (Robledo et al. 2022a). The runtime for DGN and QWINN were 16.959 and 27.587 (in seconds), respectively. We present relative error plots and another elementary example in the supplementary material.

5 Application in Scheduling

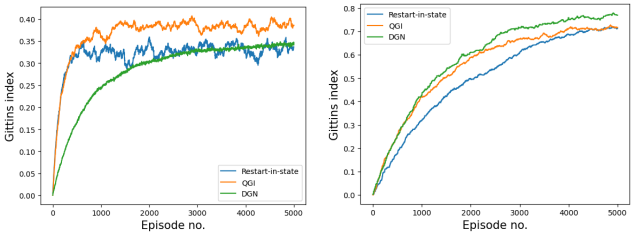
Consider a system with a single server and a given set of K jobs available at time 0. The service times of these jobs are sampled from an arbitrary distribution denoted by $F(\cdot)$. Jobs are preemptive, i.e., their service can be interrupted and resumed at a later time. For any job scheduling policy π, let T_i denote the completion time of job i and $L^\pi := \sum_{i=1}^{K} T_i$ denote the flowtime. A scheduling policy that minimizes the mean flowtime is the the Gittins index based scheduling policy (Gittins, Glazebrook, and Weber 2011; Pinedo 2012). To model this problem as an MAB, note that, each job corresponds to an arm and the current state of an arm is the amount of service that the job has received till now (known as the age of the job). There is no reward from pulling an arm (serving a job) untill it has received complete service. We receive a unit reward once the required service is received. See (Gittins, Glazebrook, and Weber 2011; Pinedo 2012) for more details on the equivalence between the flowtime minimization problem and the MAB formulation as stated above. We are particularly interested in the setting of this problem where the job size distribution $F(\cdot)$ is not known and a batch of K jobs are made available episodically. We want to investigate if the proposed algorithms QGI and DGN are able to learn the optimal Gittins index based job scheduling policy when the service time distributions $F(\cdot)$ are unknown. We first consider the case where job have service time distributions with either constant or monotonic hazard rates. We also consider other discrete distributions (Poisson, Geometric and Binomial) and illustrate the empirical episodic regret from using our algorithm. In supplementary material we also consider scheduling with continuous job size distributions where each pull of an arm corresponds to giving a fixed quanta of service (denoted by Δ). This allows us to discretize the problem for efficient implementation. More specifically, we consider uniform and log-normal service time distribution for our experiments. For a vanishingly small choice of Δ, the state space (possible ages a job can have) becomes arbitrarily large which is when the space saving advantage of our algorithms become more apparent. Note that in all the experiments below, we found QWI and QWINN to have a very poor performance in the scheduling problem due to their sensitivity to chosen learning rates (refer to supplementary material). We therefore compare our results with only that of (Duff 1995).

Discrete service time distribution

We first consider the case when the job size distribution are discrete and have a possibly state dependent parameter as described below. Each of the K jobs represent an arm which start each episode in state 1 which denotes that no service has yet been given. When an arm i (read job i) in state $s_n(i)$ (with age $s_n(i)$) is picked/served at the n^{th} step, the agent observes a transition either to $s_{n+1}(i) = s_n(i)+1$ of service with probability $1 - \rho^{s_n(i)}(i)$ or to $s_{n+1}(i) = 0$ with probability $\rho^{s_n(i)}(i)$ if the job is completed. Upon completing a job the agent receives a reward of 1, zero otherwise. We assume that jobs that have completed service leave the system and are not available. In the appendix, we illustrate an alternate approach where completed jobs are not removed and can be chosen for service, in fact resulting in service being wasted. To minimize such wasted service efforts, we heavily penalize the action of serving a finished job. We note that the results from both formulations are identical.

We first consider the case when jobs have a Geometric distribution with constant hazard rate parameter $\rho(i)$ for the i^{th} job. Clearly, jobs are heterogeneous in nature and their transition probabilities are state independent and satisfy $\rho^{s(i)}(i) = \rho(i)$. In our experiments, we chose 10 jobs with corresponding $\rho(i)$'s sampled uniformly from the unit interval $[0, 1]$ before starting the experiment. We set $\gamma = 0.99$ and epsilon is set to 1 initially and decays as $\epsilon_{n+1} = \epsilon_n \times 0.9995$. For this policy, we performed grid search across learning rates. The final obtained hyperparameters for all the experiments in this section are presented in supplementary material for reproducibility.

Here the optimal scheduling policy is known to choose jobs in the order of decreasing hazard rates. We compare the convergence behavior of our algorithm with that of (Duff 1995) for a particular state 1 and a particular arm (job 6) in Fig. 3. We observe that QGI rises more faster than restart-in-state while DGN indices converge more smoothly before settling to the optimal value. The runtimes were 21.809, 22.705 and 1218.43 (in seconds) for QGI, restart-in-state and DGN, respectively. The runtime table for all experi-



(a) (b)

Figure 3: Performance comparison of the QGI, DGN and Restart-in-state for: (a) Constant hazard rates and (b) Increasing hazard rates

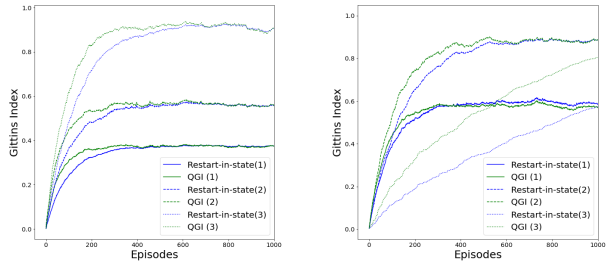
ments is presented in supplementary material.

Next, we consider the case of monotonically increasing hazard rates. As in (Duff 1995), we consider jobs where the hazard rate changes monotonically as the job receives service. Furthermore, these hazard rate parameters are updated in an exponential fashion using a parameter λ. In our experiment we assume there are 9 jobs, i.e., $K = 9$ and the maximum state is 49, i.e, $N = 50$. Before starting the trials, we sample the initial hazard rates ($\rho^{(1)}(i)$ for all i) uniformly from $[0, 1]$. See supplementary material for equations governing the hazard rate and detailed discussion on the decreasing hazard rate case. We set $\lambda = 0.8$, $\gamma = 0.9$ and $\epsilon_{n+1} = \epsilon_n \times 0.9995$. Again, we ran our proposed algorithms with tuned learning rates for the case where all jobs have an increasing hazard rate. Note that the scheduling policies learnt by our algorithm coincide with the known optimal policy of FIFO in this setting. For a performance comparison among the tabular and Deep RL methods, we consider the convergence of state 4 of job 4. While the indices flatline by 7000 episodes, plots are restricted to 5000 episodes to show the evolution of Gittins index. Results are presented in Fig. 3.

Lastly, we consider Binomial(n, p), Poisson(Λ) and Geometric(q) job size distributions. We consider a batch of 4 jobs in every episode. When an arm i (read job i) in state $s_n(i)$ (with age $s_n(i)$) is picked/served at the n^{th} step, the server observes a transition to $s_{n+1}(i) = s_n(i) + 1$ as long as $\tau_i \neq s_n(i)+1$. If $\tau_i = s_n(i)+1$, then the job has received complete service and is removed from the system.

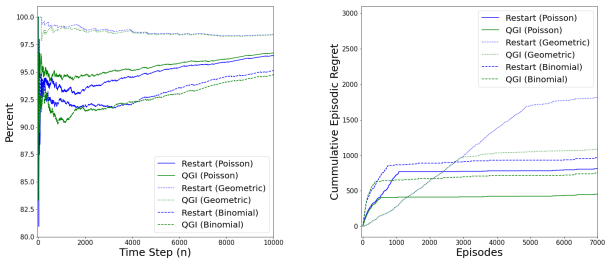
In our experiment with the three distributions we choose the parameters $n = 10, p = 0.5, \lambda = 5$ and $q = 0.5$. We set $\gamma = 0.99$ and epsilon is set to 1 initially an decays as $\epsilon_{n+1} = \epsilon_n \times 0.9995$. In Fig. 4 we compare the performance of tabular methods (for states 1,2 and 3) for Binomial and Poisson distribution. The convergence results for geometric distribution are presented in supplementary material. While the speed of convergence depends on the type of distribution considered and the corresponding states, QGI seems to outperform restart-in-state for all states and all distributions.

In Fig. 5, we plot the percentage of time QGI and restart-in-state chose the optimal action as a function of the iteration number for the three distributions. Here again QGI displays a better accuracy, especially for the Poisson distribution. We



(a) (b)

Figure 4: Convergence of Gittins Indices for (a) Binomial and (b) Poisson service time distributions.



(a) (b)

Figure 5: (a) shows the % of steps for which an optimal action is performed while (b) shows the cumulative episodic regret collected over trials.

also plot the cumulative episodic regret for the two algorithms (for all the three distributions). Here the episodic regret was calculated as the difference in the flowtime between QGI and the optimal Gittins index policy in an episode. Again, we see that the cumulative regret for QGI is lower than Restart-in-state, clearly demonstrating the advantages of our method.

6 Conclusion

In this work, we have introduced QGI and DGN which are tabular and Deep RL based methods for learning the Gittins indices using the retirement formulation. To illustrate the applicability of our method, we consider the problem of learning the optimal scheduling policy that minimizes the mean flowtime for batch of jobs with arbitrary but unknown service time distributions. Through our experiments, we have shown that our methods have better convergence performance, require less memory and also offer lower empirical cumulative regret. There are several open directions that beg further investigation. We would also like to investigate the applicability of our method to learning the optimal scheduling policy in an $M/G/1$ queue minimizing the mean sojourn time. While the algorithms that we propose and those that are in the literature are essentially value function based, we would also like to see if a policy gradient based approach can be used to learn the Gittins index.

References

Aalto, S.; Ayesta, U.; and Righter, R. 2009. On the Gittins index in the M/G/1 queue. *Queueing Systems*, 63: 437–458.

Aalto, S.; and Scully, Z. 2023. Minimizing the mean slowdown in the M/G/1 queue. *Queueing Systems*, 104(3): 187–210.

Akbarzadeh, N.; and Mahajan, A. 2023. On learning Whittle index policy for restless bandits with scalable regret. *IEEE Transactions on Control of Network Systems*.

Avrachenkov, K. E.; and Borkar, V. S. 2022. Whittle index based Q-learning for restless bandits with average reward. *Automatica*, 139: 110186.

Duff, M. O. 1995. Q-learning for bandit problems. In *Machine Learning Proceedings 1995*, 209–217. Elsevier.

Frostig, E.; and Weiss, G. 2016. Four proofs of Gittins' multiarmed bandit theorem. *Annals of Operations Research*, 241(1-2): 127–165.

Gast, N.; Gaujal, B.; and Yan, C. 2023. Exponential asymptotic optimality of Whittle index policy. *Queueing Systems*, 1–44.

Gittins, J. 1974. A dynamic allocation index for the sequential design of experiments. *Progress in statistics*, 241–266.

Gittins, J.; Glazebrook, K.; and Weber, R. 2011. *Multi-armed bandit allocation indices*. John Wiley & Sons.

Gittins, J. C. 1979. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 41(2): 148–164.

Lattimore, T.; and Szepesvári, C. 2020. *Bandit algorithms*. Cambridge University Press.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Nakhleh, K.; Hou, I.; et al. 2022. DeepTOP: Deep threshold-optimal policy for MDPs and RMABs. *Advances in Neural Information Processing Systems*, 35: 28734–28746.

Pinedo, M. L. 2012. *Scheduling*, volume 29. Springer.

Robledo, F.; Borkar, V.; Ayesta, U.; and Avrachenkov, K. 2022a. QWI: Q-learning with Whittle index. *ACM SIGMETRICS Performance Evaluation Review*, 49(2): 47–50.

Robledo, F.; Borkar, V. S.; Ayesta, U.; and Avrachenkov, K. 2022b. Tabular and deep learning of Whittle index. In *EWRL 2022-15th European Workshop of Reinforcement Learning*.

Scully, Z. 2023. A New Toolbox for Scheduling Theory. *ACM SIGMETRICS Performance Evaluation Review*, 50(3): 3–6.

Scully, Z.; Grosof, I.; and Harchol-Balter, M. 2020. The Gittins policy is nearly optimal in the M/G/k under extremely general conditions. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(3): 1–29.

Tsitsiklis, J. N. 1994. A short proof of the Gittins index theorem. *The Annals of Applied Probability*, 194–199.

Verloop, I. M. 2016. Asymptotically optimal priority policies for indexable and nonindexable restless bandits. *The Annals of Applied Probability*, 26(4): 1947—-1995.

Weber, R. R.; and Weiss, G. 1990. On an index policy for restless bandits. *Journal of applied probability*, 27(3): 637–648.

Whittle, P. 1980. Multi-armed bandits and the Gittins index. *Journal of the Royal Statistical Society: Series B (Methodological)*, 42(2): 143–149.

Whittle, P. 1988. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 25(A): 287–298.