Supplementary Material for Tabular and Deep Reinforcement Learning for Gittins Index

Anonymous

1 Related works

1.1 The restart-in-state formulation

Duff was the first to propose a Q-learning based RL algorithm to learn the Gittins' indices for the MAB problem [3]. The algorithm is based on an equivalent formulation for the MAB problem where for any fixed arm, the agent has a restart action that first teleports it to a fixed state x instantaneously before moving to the next state. We henceforth call this algorithm as the restart-in-state algorithm. Since the discussion below mostly concerns a fixed arm, we drop the superscript i to denote the arm for notational convenience. Instead, we denote corresponding optimal value function from starting in state j by $V^x(j)$ (the super script x now denotes the state that you will always teleport to via the restart action 0) and is given by $V^x(j) = \max\{Q^x(j,1), Q^x(j,0)\}$ where $Q^x(j,1) = r(j,1) + \gamma \sum_k p(k|j,1)V^x(k)$ and $Q^x(j,0) = r(x,1) + \gamma \sum_k p(k|x,1)V^x(k)$.

Here, $Q^x(j,1)$ denotes the state action value function for continuing in state j while $Q^x(j,0)$ denotes the value corresponding to the restart action. Note that when the restart action is chosen in state j, there is an instantaneous transition to state x at which point the action 1 is performed automatically resulting in the immediate reward of $r(x,1)$ to be earned. For some more notational convenience we will now (and occasionally from here on) denote $s_n(i)$ by s_n and $r^i(s_n(i),1)$ by $r(s_n)$, respectively. We will also suppress the action dimension, and the superscript representing arm i in the reward notation when the context is clear. Given an MAB with K (homogeneous) arms with N states per arm, the agent pulls an arm i in the n^{th} step via a Boltzman distribution and observes a state transition from s_n to

s_{n+1} and receives a reward $r(s_n)$. For learning the Gittins index using a Q learning approach, this single transition is utilised to do two updates for each $k \in \mathcal{S}$. The first update corresponds to taking the action 'continue' in state s_n and transitioning to state s_{n+1} for all restart in k problems. The second update corresponds to choosing to teleport to s_n from any state k and then observing the transition to s_{n+1}. A single reward $r(s_n)$ leads to the following $2N$ Q-learning updates where for each $k \in \mathcal{S}$ we perform:

$$Q^k_{n+1}(s_n, 1) = (1 - \alpha(n))Q^k_n(s_n, 1) + \alpha(n)\left[r(s_n) + \gamma \max_{a\in\{0,1\}} Q^k_n(s_{n+1}, a)\right]$$

$$Q^{s_n}_{n+1}(k, 0) = (1 - \alpha(n))Q^{s_n}_n(k, 0) + \alpha(n)\left[r(s_n) + \gamma \max_{a\in\{0,1\}} Q^{s_n}_n(s_{n+1}, a)\right]$$

Here, $\alpha(n)$ is the learning rate and the Gittins index for a state x is tracked by the $Q^x_n(x, 1)$ value. In writing the preceeding two equations, we are assuming that the arms are homogeneous and therefore the same Q-values can be updated, irrespective of the arm chosen. For a more general setting where the arms can be heterogeneous, we need to have a separate Q-table for each arm. Needless to say, the time complexity for each iteration is $O(2N + N)$ and space complexity is $O(2 \cdot N^2 \cdot K)$. See [3] for more details.

1.2 A Whittle index approach to learn Gittins index

The Whittles index is a heuristic policy introduced in [6] for the restless multi-arm bandit problem (RMAB). Here, M out of K arms must be pulled and the passive arms (arms that are not pulled) are allowed to undergo state transitions. For the setting where state transitions probabilities are unknown (for both active and passive arms), RL based methods (QWI, QWINN) to learn the Whittle indices have recently been proposed [4, 5]. When the passive arms do not undergo state transitions, the Whittle index coincides with the Gittins index and therefore these algorithms can in fact be used to learn the underlying Gittins index. Using the notion of a reference state x (see [4] for details), the corresponding Q-learning based update equations for learning the Gittins index are as follows:

$$\begin{aligned} Q^x_{n+1}(s_n, a_n) &= (1-\alpha(n))Q^x_n(s_n, a_n) + \alpha(n)\Big((1-a_n)\lambda_n(x) \\ &+ a_n r(s_n) + \gamma \max_{v\in\{0,1\}} Q^x_n(s_{n+1}, v)\Big) \end{aligned}$$

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

Here, $\lambda_n(x)$ denotes the Whittles index for state x and $r(s_n)$ denotes the reward for pulling an arm in state s_n, in the n^{th} step. As earlier, the dependence on arm i is suppressed in the notation.

2 Proof outline for Theorem 1

The proof follow along similar lines to that in [5]. Also see [2] for similar results. First, let us recall the update rules used in QGI from the main text:

$$Q^x_{n+1}(s_n, 1) = (1-\alpha(n))Q^x_n(s_n, 1) + \alpha(n)\left(r(s_n) + \gamma \max\{Q^x_n(s_{n+1}, 1), M_n(x)\}\right) \tag{1}$$

and

$$M_{n+1}(x) = M_n(x) + \beta(n)\left(Q^x_{n+1}(x,1) - M_n(x)\right). \tag{2}$$

Consider the following equations which will be compared to equations (1) and (2) later:

$$x_{n+1} = x_n + a(n)\left[h(x_n, y_n) + L^{(1)}_{n+1}\right] \tag{3}$$

$$y_{n+1} = y_n + b(n)\left[g(x_n, y_n) + L^{(2)}_{n+1}\right] \tag{4}$$

In these equations, the functions h and g are continuous Lipschitz functions, the L_n are martingale difference sequences representing noise terms, and $a(n)$ and $b(n)$ are step-size terms satisfying $\sum_n \alpha(n) = \infty, \sum_n \alpha(n)^2 < \infty$, $\sum_n \beta(n) = \infty, \sum_n \beta(n)^2 < \infty$ and $\frac{b(n)}{a(n)} \to 0$ as $n \to \infty$. These conditions are necessary for stable convergence of (3) and (4). Note that, for notational convenience, we use $M_x(n)$ to denote the value of Gittins estimate of state x in n^{th} iteration. Here, (3) represents the Q-learning update as in (1) and (4) represents equation the stochastic approximation step to update the indices as in (2). First, let us define $F^M_s(\Psi(j,b))$ and $L_{n+1}(s)$ such that:

$$F^M_s(\Psi(j,b)) = R(s) + \gamma \sum_j p(j \mid i, u) \max\{\Psi(j,1), M_j\}$$

$$L_{n+1}(s) = R(s) + \max\left\{Q_n^x\left(s_{n+1}, 1\right), M_n(x)\right\} - F_s^{M_x(n)}\left(Q_n\right).$$

Using these, we can rewrite the equation (1) as:

$$Q_{n+1}^x(s,u) = Q_n^x(s,u) + \alpha(n)\left[F_s^{M_x(n)}\left(Q_n\right) - Q_n^x(s,u) + L_{n+1}(s)\right] \tag{5}$$

Comparing equations (3) and (5) we can make the correspondence $a(n) = \alpha(n), h\left(x_n, y_n\right) = F_s^{M_x}(n)\left(Q_n\right) - Q_n$, where $x_n = Q_n$, $y_n = M(n)$ and $L_{n+1}(s)$ is the martingale difference sequence $L_{n+1}^{(1)}$. Equations (2) and (4) correspond to $b(n) = \beta(n)$, $g\left(x_n, y_n\right) = Q_n^x(x,1) - M_x(n)$ and a martingale difference sequence $L_{n+1}^{(2)} = 0$. Now let $\tau(n) = \sum_{m=0}^{n} \alpha(m), m \geq 0$. Define $\bar{Q}(t), \bar{\lambda}(t)$ as the interpolation of the trajectories of Q_n^x and $M_x(n)$ on each interval $[\tau(n), \tau(n+1)], n \geq 0$ as:

$$\bar{Q}(t) = Q(n) + \left(\frac{t - \tau(n)}{\tau(n+1) - \tau(n)}\right)\left(Q(n+1) - Q(n)\right)$$
$$\bar{M_x}(t) = M_x(n) + \left(\frac{t - \tau(n)}{\tau(n+1) - \tau(n)}\right)\left(M_x(n+1) - M_x(n)\right)$$
$$t \in [\tau(n), \tau(n+1)]$$

which track the asymptotic behavior of the coupled o.d.e.s

$$\dot{Q}(t) = h(Q(t), M_x(t)), \dot{M_x} = 0$$

Here the latter is a consequence of considering the following form of equation (2) and putting $\frac{\beta(n)}{\alpha(n)} \to 0$:

$$M_x(n+1) = M_x(n) + \alpha(n)\left(\frac{\beta(n)}{\alpha(n)}\right)\left(Q_n^x(x,1) - M_x(n)\right)$$

Now, $M_x(\cdot)$ is a constant of value M_x' for $Q(t)$ due to $M_x(\cdot)$ being updated on a slower time scale. Because of this, the first o.d.e. becomes $\dot{Q} = h\left(Q(t), M_x'\right)$, which is well posed and bounded, and has an asymptotically stable equilibrium at Q_M^* [Theorem 3.4, [1]]. This implies that $Q_n^x - Q_{M_x(n)}^* \to 0$ as $n \to \infty$. On the other hand, for $M_x(t)$, let us consider a second trajectory on the second time scale, such that:

$$\tilde{M_x}(t) = M_x(n) + \left(\frac{t - \tau'(n)}{\tau'(n+1) - \tau'(n)}\right)\left(g(n+1) - g(n)\right)$$
$$t \in [\tau'(n), \tau'(n+1)], \tau'(n) = \sum_{m=0}^{n} \beta(m), n \geq 0$$

which tracks the o.d.e.:

$$\dot{M}_x(t) = Q^*_{M_x(t)}(x, 1) - M_x(t)$$

As in the previous case this bounded o.d.e converges to an asymptotically stable equilibrium where M satisfies $Q^*_M(x, 1) = M_x$. This is the point of indifference between retiring and continuing, and hence, M characterises the Gittins index.

3 Sensitivity to hyperparameters

In our initial experiments, both for elementary examples as well as for scheduling, we observed that QWI was very sensitive to the learning rates chosen. Upon observing the step wise updates to Q-values and Whittles estimates in various problem settings, we hypothesise that the reason for this is the noise introduced by the Whittle index term while updating the Q-value for passive action for every passive arm. This creates a positive self-reinforcing cycle of subsequent updates causing the values of Whittle estimates and associated Q-values to blow up for a state unless the updates are governed by the right set of learning rates for the given problem setting. Say, for some multi-arm bandit problem with 10 homogeneous arms, each having 10 states, in any n^{th} step, the number of updates to $Q^k_n(s_n(i), 1)$ values are 10 while 90 updates are done to $Q^k_n(s_n(i), 0)$ values for all $k \in N$, $i \in K$. These passive Q-values are used in updates to $Q^k_n(k, 1)$, which directly impacts the update to Whittles estimate $W(k)$ for any $k \in N$. In contrast, QGI doesn't use the $Q^k_n(s_n(i), 0)$ terms as they are replaced by $M_n(k)$ itself in updates to other Q-values, making it empirically stabler.

For the elementary problem discussed in main paper, we decided to tune the learning rates based on the structure presented in [4]. Hence, we perform a grid search across the hyperparameters x, y, θ, κ and ϕ defined as follows: $\alpha(n) = \frac{x}{\lceil \frac{n}{\theta} \rceil}$, $\beta(n) = \frac{y}{1+\lceil \frac{n \log n}{\kappa} \rceil} I\{n \mod \phi \equiv 0\}$. In a grid of 4200 hyperparameter combinations considered, we found Gittins indices in 227 combinations to have converged in a $\delta = 0.02$ neighbourhood of the true values for QGI, while this was true for only 9 combinations in QWI. We chose the final learning rates used in the convergence plots shown in the main paper from this refined set by choosing the combination with the lowest sum of absolute difference between true and empirical Gittins estimates for each

state. The empirical Gittins index was estimated by taking the average of last 200 values obtained during training. It is also important to note that we observed problems in reproducibility of convergence for a fixed set of learning rates in QWI, which was accounted for by performing multiple (100) runs for each refined hyperparameter combination and choosing the one with highest rate of convergence during the tuning process in addition to the absolute difference based criterion.

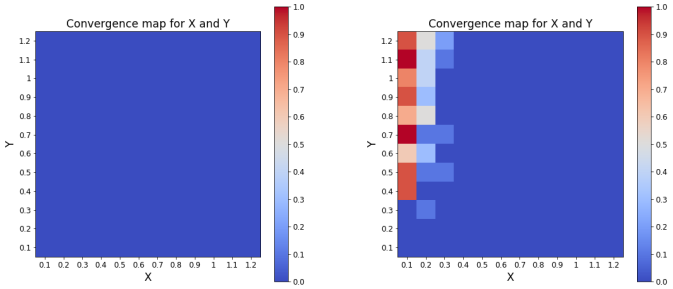
To analyze the sensitivity of QWI and QGI to the hyperparameters of the learning rates, for each chosen hyperparameter, we plot the proportion of times the empirical Gittins estimates converged to within a δ-neighborhood of the true values (for different δ values of $0.01, 0.025, 0.05$) for all states of the elementary example. To illustrate the sensitivity of QWI to its hyperparameters, we vary two hyperparameters across a grid at a time, giving us a 2D heatmap that we call the "Convergence Map". The convergence maps for different δ-neighbourhoods with respect to changes in hyperparameter pairs $\{x, y\}$, $\{\theta, \kappa\}$, and $\{\phi, \mathbf{y}\}$ are shown in the Fig 1, 2 and 3, respectively.

The algorithms were run 10 times for each hyperparameter setting, with each run being 20,000 time steps long. Clearly, QGI is more robust with respect to the hyperparameter combination chosen (indicated by the larger non blue region in all plots compared to QWI), and the results are more reproducible for the same set of chosen learning rates (indicated by more number of warmer zones). Across our experiments, this translated to general robustness of QGI for any chosen set of learning rates, irrespective of their structure or update rules. An implication of this effect was that we could not find the right set of hyperparameters for QWI in the scheduling problem discussed in main paper and hence decided to leave out QWI in the convergence and runtime comparisons drawn there. The same holds true for the application of QWINN for scheduling as well.

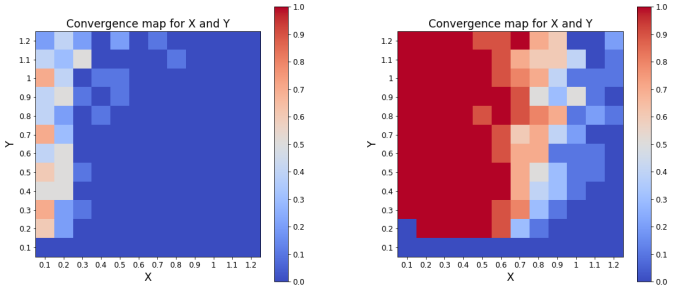
4 Another elementary example

First, let us consider the elementary problem discussed in the main text. Along with the convergence plots presented, we now plot the evolution of 'percent of wrong actions taken' vs 'time step'. The result is presented in Fig. 4. Clearly, QGI chooses the least % of wrong actions during most part of training under the same epsilon policy to choose actions as others.

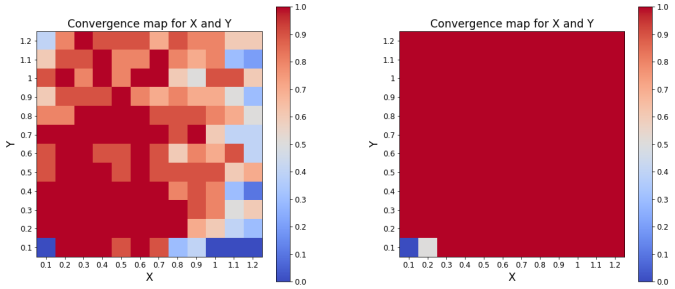
Now, let us discuss another elementary example which was considered by



(a) $\delta = 0.01$

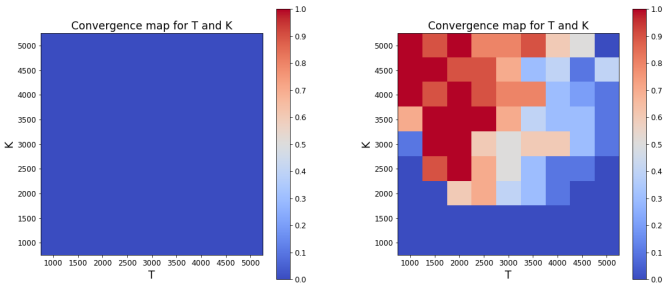


(b) $\delta = 0.025$

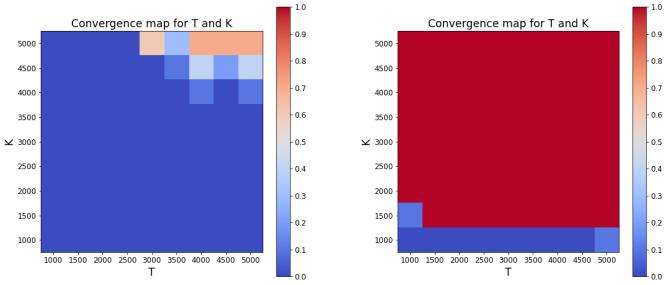


(c) $\delta = 0.05$

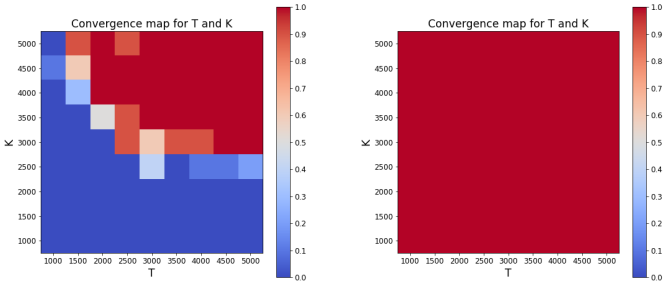
Figure 1: Comparison of convergence for QWI (left) and QGI (right) across different zones for $\{x, y\}$.



(a) $\delta = 0.01$

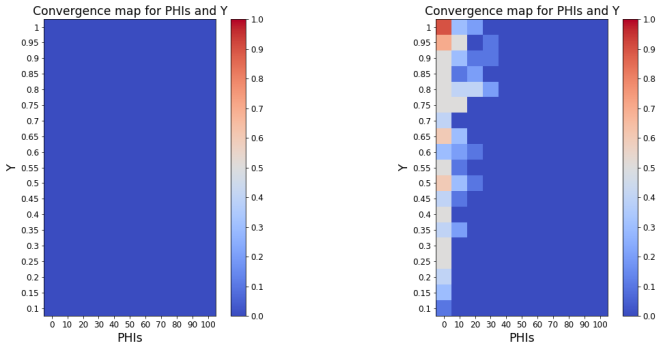


(b) $\delta = 0.025$

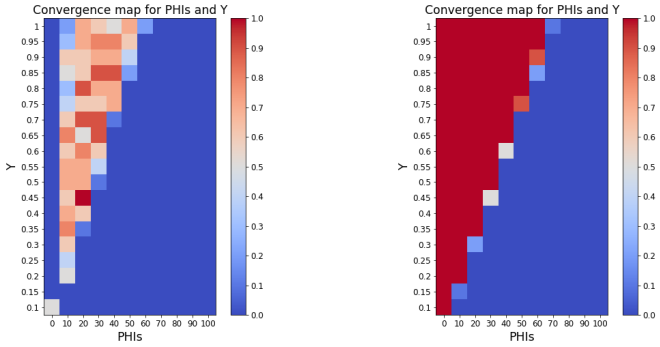


(c) $\delta = 0.05$

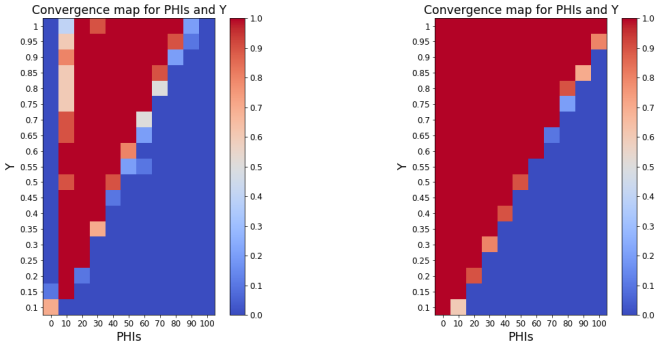
Figure 2: Comparison of convergence for QWI (left) and QGI (right) across different zones for $\{\theta, \kappa\}$.



(a) $\delta = 0.01$



(b) $\delta = 0.025$



(c) $\delta = 0.05$

Figure 3: Comparison of convergence for QWI (left) and QGI (right) across different zones for $\{\phi, \mathbf{y}\}$.

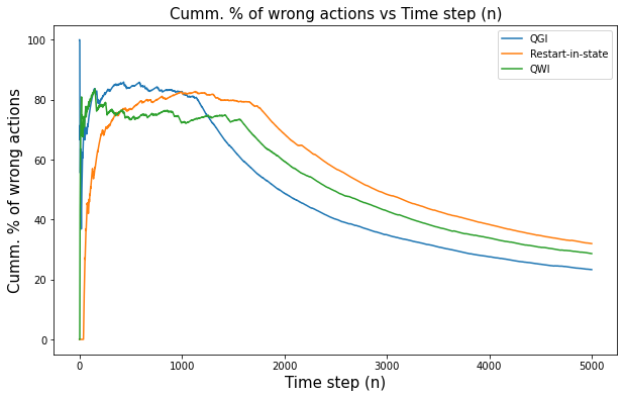


Figure 4: Plot of % wrong actions taken for elementary problem

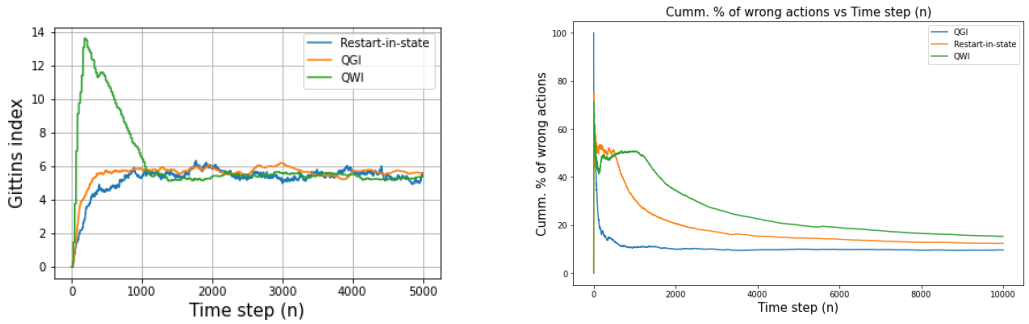
Duff in the restart-in-i paper [3]. The problem is relatively simpler. There are two heterogeneous arms, with states 0 and 1 in each arm. The transition matrices for the two arms (P_{arm-0}, P_{arm-1}) and the reward matrix (R) is as follows:

$$P_{arm-0} = \begin{bmatrix} 0.3 & 0.7 \\ 0.7 & 0.3 \end{bmatrix}$$

$$P_{arm-1} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}$$

$$R = R_{arm-0} = R_{arm-1} = \begin{bmatrix} 1 & 10 \\ 1 & 10 \end{bmatrix}$$

Under $\gamma = 0.9$ and $\epsilon = 0.2$, we ran the mentioned experiment on all three tabular algorithms. The hyperparameters were kept same as the elementary problem in main text. Note that we leave out neural networks for this example due to the simplicity of the environment dynamics. We get the convergence results and % wrong actions taken as shown in Fig 5. In this elementary example as well we found QGI to be more stable than QWI and rise more quickly to the true index than Restart-in-state.



(a) (b)

Figure 5: (a) Convergence of State 1 of Arm 0 and (b) Plot of % wrong actions taken.

5 Runtime discussion

One of the main highlighted advantages of our proposed algorithm QGI and deep learning counterpart DGN is the space saving that comes with the algorithms. Recall that we do not need to store the Q-values corresponding to action 0 in QGI and hence save the time taken in performing updates to those Q-values. The number of updates per step is shown for all discussed algorithms in Table 3. In DGN, the replay buffer does not store the action 0 tuples and we only need to do N learning step once for each step in the environment as compared to QWINN where $N \cdot K$ updates are done as passive arms are not ignored. These differences in the fundamental structure of algorithms result in better real-time runtime for QGI and DGN in our experiments as shown in Table 1 and 2.

The experiments were run on a local machine with the following hardware specifications: AMD Ryzen 7 5800H, 16 GB RAM, NVIDIA GeForce RTX 3050 (4 GB VRAM). The codes were compiled with Python 3.10.13. While convergence time is marginally better for the toy problem due to a small environment, runtime advantage is clearly noticeable in scheduling (especially continuous case) and DQN based algorithms. It is also noteworthy that the implications of space saving were also evident while running the codes for obtaining convergence maps. For the code to obtain the $\{x, y\}$ pair plots, the QWI code ran for a total of 1 hour 06 minutes while QGI only took 24 minutes 37 seconds.

Algorithm	Toy Problem	Constant HR	Inc. HR	Dec. HR	Uniform	Log-norm
QGI	0.20927	21.809	28.646	36.449	993.194	2109.424
Restart-in	0.30008	22.705	29.509	43.564	1072.586	2142.650
QWI	0.68615	-	-	-	-	-

Table 1: Comparison of runtime for Tabular Methods (in seconds)

Algorithm	Toy Problem	Constant HR	Inc. HR	Dec. HR
DGN	16.959	1489.43	579.964	853.711
QWINN	27.587	-	-	-

Table 2: Comparison of runtime for Neural networks (in seconds)

Algorithm	Homogeneous Setting	Heterogeneous Setting
1. Restart-in-i	2N	2N
2. QGI a) Q-values b) Index	 N $N\mathbb{I}_{\beta\neq 0}$	 N $KN\mathbb{I}_{\beta\neq 0}$
3. QWI a) Q-values b) Index	 KN $N\mathbb{I}_{\beta\neq 0}$	 KN $KN\mathbb{I}_{\beta\neq 0}$

Table 3: Comparison of Algorithms in Different Settings

6 Supplementary material for scheduling

6.1 Hazard rate equations

Let τ_i denote the random service time for the i^{th} job. We assume that the disribution for τ_i satisfies the following :
a) For increasing hazard rate:

$$\Pr\{\tau_i = s\} = \rho^s(i) \prod_{k=1}^{s-1} \left(1 - \rho^{(1)}(i)\right) \lambda^{k-1} \text{ where}$$

$$\rho^s(i) = \left\{1 - \left[\left(1 - \rho^{(1)}(i)\right) \lambda^{(s(i)-1)}\right]\right\}$$

b) For decreasing hazard rate:

$$\Pr\{\tau_i = s\} = \rho^s(i) \prod_{k=1}^{s-1} \left(1 - \rho^{(1)}(i)\right) \lambda^{1/(k-1)} \text{ where}$$

$$\rho^s(i) = \left\{1 - \left[\left(1 - \rho^{(1)}(i)\right) \lambda^{1/(s(i)-1)}\right]\right\}$$

These equations govern how jobs dynamically interact with the server and get served.

6.2 Decreasing hazard rate

In this subsection, we consider the case of monotonically decreasing hazard rates. This follows from the discussion on increasing hazard rates in the main text. As in [3], 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 $[0, 1]$. We set $\lambda = 0.8$, $\gamma = 0.9$ and $\epsilon_{n+1} = \epsilon_n \times 0.9995$. Note that the scheduling policies learnt by our algorithm coincide with the known optimal policy of Round-Robin in this setting. For a performance comparison among the tabular and Deep RL methods, we consider the convergence of state 4 of job 4. Results are truncated to 5000 episodes to show evolution of Gittins index and presented in Fig. 6.

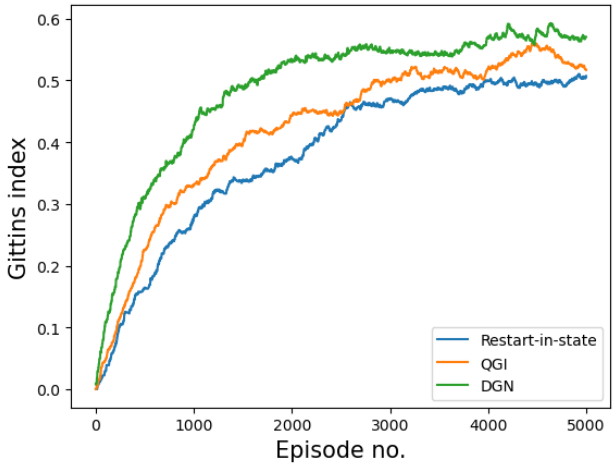


Figure 6: Performance comparison of the QGI, DGN and Restart-in-state for decreasing hazard rate.

6.3 Arbitrary service time distribution (Geometric)

This subsection directly follows from the part of section 5 where we consider Binomial(n, p), Poisson(Λ) and Geometric(q) job size distributions. Here, we present the convergence results for the geometric distribution in Fig 7.

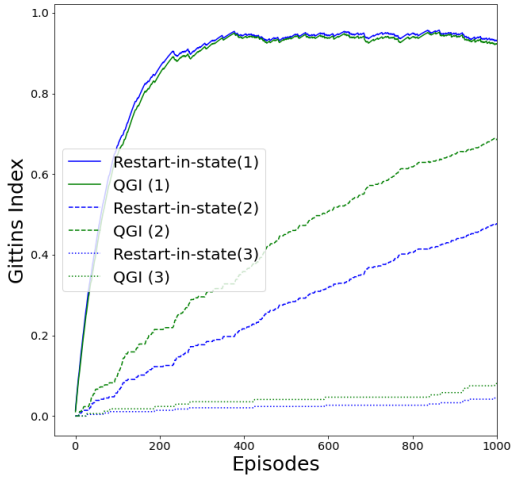


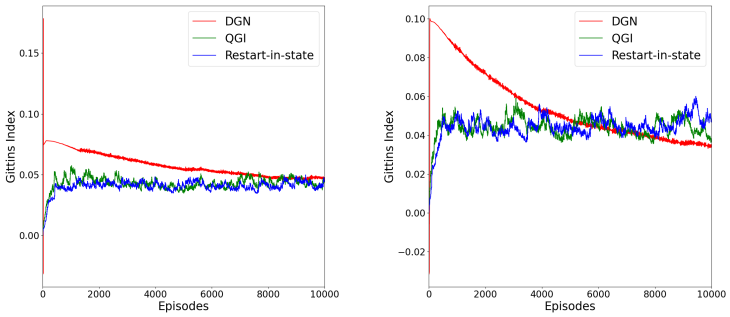
Figure 7: Performance comparison of the QGI and Restart-in-state for geometric job size distribution.

6.4 Continuous service time distribution

We now consider some continuous service time distribution for job sizes. For our experiment we consider a batch of 4 jobs per episode. Service is provided in a fixed quanta of size Δ that allows us to discretize the problem and use the existing machinery. Once a Δ is fixed, we have a discrete time MAB where the age of jobs are discrete multiples of Δ. As in monotonic hazard rate case, when an arm i in state $s_n(i)$ 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 $\tau_i \neq s_n(i) + 1$.

We first consider the experiment where job sizes are Uniform over the support $[0, 10]$ and a service quantum of size $\Delta = 0.1$. This makes the number of possible states in the system equal to 100. There are 4 jobs in each episode to schedule and the discount factor $\gamma = 0.99$ is considered. Epsilon was set

to 0.1 throughout training. The performance results are shown in Fig. 8 for state 34 (age 3.4). It can be seen that both the tabular methods have more variability as compared to DGN which converges more smoothly. We next assume that the job sizes are sampled from a log-normal distribution with parameter $\mu = log(30), \sigma = 0.6$ and $\Delta = 0.5$. We truncate the max job size to 75 (as $P(X > 75) = 0.06336$). Here, again four homogeneous jobs are taken per episode and the discount factor is $\gamma = 0.99$. Here, we set epsilon to $\epsilon = 1$ and update it as follows: $\epsilon = max(\epsilon * 0.999, 0.1)$. Note that the number of states, $|S| = 150$. Due to this, the number of Q-values being tracked in QGI are 22500 while in other algorithms 45000 cells are being filled. This leads to significant runtime advantage as shown in Table 1. The convergence results for a particular state 49 (age 24.5) are shown in Fig. 8.



(a) (b)

Figure 8: Performance comparison of the QGI, DGN and Restart-in-state for: (a) Uniform distribution and (b) Log normal distribution.

6.5 Hyperparameters

Let us discuss the hyperparameters selection for scheduling results in the main paper. Firstly, we fixed an epsilon policy based on the lesser mean flowtime. For the policy, we did a grid search across learning rates to obtain the final values used to plot the figures presented in the main paper. Let us now discuss the criterion for assessing different runs obtained from the grid search. Since we did not know the true estimates of Gittins indices for a given job (arm) in a certain age (state), we chose the final hyperparameters from the grid search based on the following criterion:

- The learnt Gittins index policy is consistent with the hazard rate distribution across states and arms.
- The convergence is smooth and stable.
- The flowtime is minimised.

These values are presented in Table 4 and 5.

Condition	**Restart-in-state**	**QGI**
Constant Hazard Rate	$\alpha(n) = 0.2$	$\alpha(n) = 0.6$, $\beta(n) = 0.4 \times 1_{\{n \text{ modulo } 5 \equiv 0\}}$
Monotonic Hazard Rate	$\alpha(n) = 0.3$	$\alpha(n) = 0.6$, $\beta(n) = 0.4 \times 1_{\{n \text{ modulo } 5 \equiv 0\}}$
Arbitrary Distributions	$\alpha(n) = 0.3$	$\alpha(n) = 0.6$, $\beta(n) = 0.3 \times 1_{\{n \text{ modulo } 2 \equiv 0\}}$

Table 4: Hyperparameters for tabular methods for scheduling

Condition	**DGN**
Constant Hazard Rate	LR = 5e-2, BATCH_SIZE = 64, $\beta(n) = 0.3 \times 1_{\{n \text{ modulo } 2 \equiv 0\}}$
Monotonic Hazard Rate	LR = 5e-3, BATCH_SIZE = 32, $\beta(n) = 0.5 \times 1_{\{n \text{ modulo } 15 \equiv 0\}}$

Table 5: Hyperparameters for DGN for scheduling

7 Reproducibility checklist

- Includes a conceptual outline and/or pseudocode description of AI methods introduced: **Answer**: yes
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results: **Answer**: yes
- Provides well marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper: **Answer**: yes
- **Does this paper make theoretical contributions? Answer**: yes
- If yes, please complete the list below:
  - All assumptions and restrictions are stated clearly and formally. **Answer**: yes
  - All novel claims are stated formally (e.g., in theorem statements). **Answer**: yes

  - Proofs of all novel claims are included. **Answer**: yes
  - Proof sketches or intuitions are given for complex and/or novel results. **Answer**: yes
  - Appropriate citations to theoretical tools used are given.**Answer**: yes
  - All theoretical claims are demonstrated empirically to hold. **Answer**: yes
  - All experimental code used to eliminate or disprove claims is included. **Answer**: yes

- **Does this paper rely on one or more datasets? Answer**: NO

- **Does this paper include computational experiments? Answer**: yes

- If yes, please complete the list below:

  - Any code required for pre-processing data is included in the appendix. **Answer**: No, as this step is not needed given the structure of the problem discussed.
  - All source code required for conducting and analyzing the experiments is included in a code appendix. **Answer**: yes
  - All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. **Answer**: yes
  - All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from. **Answer**: partial. Comments are present detailing the implementation, but they don't include where each step comes from. We give pseudocodes for both our algorithms so that it is easy to comprehend and implement the codes by oneself.
  - If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. **Answer**: no, but since we focus on small environments and a convergence based analysis in the results, they are indifferent from exact transitions as long as the hyperparameters are same.

– This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. **Answer**: Partial. While the computing infrastructure is delineated, one benefit of our algorithm is that it does not require off-the-shelf frameworks. We use numpy, pandas and matplotlib for basic array manipulation and plotting, so we do not list their versions used.

– This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. **Answer**: yes

– This paper states the number of algorithm runs used to compute each reported result. **Answer**: no. This is because different runs produce almost identical plots as long as the hyperparameters are same.

– Analysis of experiments goes beyond single-dimensional summaries of performance to include measures of variation, confidence, or other distributional information. **Answer**: yes

– The significance of any improvement or decrease in performance is judged using appropriate statistical tests. **Answer**: yes

– This paper lists all final (hyper-)parameters used for each model /algorithm in the paper's experiments. **Answer**: yes

– This paper states the number and range of values tried per (hyper-)parameter during development of the paper, along with the criterion used for selecting the final parameter setting. **Answer**: yes

References

[1] Jinane Abounadi, Dimitrib Bertsekas, and Vivek S Borkar. Learning algorithms for markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698, 2001.

[2] Vivek S Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.

[3] Michael O Duff. Q-learning for bandit problems. In *Machine Learning Proceedings 1995*, pages 209–217. Elsevier, 1995.

[4] Francisco Robledo, Vivek Borkar, Urtzi Ayesta, and Konstantin Avrachenkov. Qwi: Q-learning with whittle index. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):47–50, 2022.

[5] Francisco Robledo, Vivek S Borkar, Urtzi Ayesta, and Konstantin Avrachenkov. Tabular and deep learning of Whittle index. In *EWRL 2022-15th European Workshop of Reinforcement Learning*, 2022.

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