Q-Learning Lagrange Policies for Multi-Action Restless Bandits

Online Appendix

A PROOF OF CONVERGENCE FOR MAIQL

In this section, we provide a detailed proof of the convergence for MAIQL. We begin by stating 2 standard assumptions for establishing the convergence guarantee of Q-learning in the average-reward setting, and then add a third that's required for two time-scale convergence.

Assumption 1 (Uni-chain Property). *There exists a state s_0 that is reachable from any other state $s \in S$ with a positive probability under any policy.*

This property formalises the notion that there aren't any 'forks' in the MDP, in each of which very different outcomes could occur. This is important because, if there were a fork, the notion of 'average' reward would be ill-defined as it would depend on which 'fork' gets taken.

Assumption 2 (Asynchronous Update Step-Size). *The sequence of step-sizes $\{\alpha(t)\}$ satisfy the following properties for any $x \in (0,1)$:*

$$\sup_t \frac{\alpha(\lfloor xt \rfloor)}{\alpha(t)} < \infty$$

$$\sup_{y\in[x,1]} \left| \frac{\sum_{m=0}^{\lfloor yt \rfloor} \alpha(m)}{\sum_{m=0}^{t} \alpha(m)} - 1 \right| \to 0$$

This is a condition that is required to show that updating $Q(s,a_j)$ values one at a time with an ϵ-greedy policy is equivalent to updating all the $Q(s,a_j)$ values together, in expectation.

Assumption 3 (Relative Step-Size). *The two sequences of step-sizes, $\{\alpha(t)\}$ and $\{\gamma(t)\}$, satisfy the following properties:*

$$\text{(A) } \textit{Fast Time-Scale:} \quad \sum_{t=0}^{\infty} \alpha(t) \to \infty, \qquad \sum_{t=0}^{\infty} \alpha^2(t) < \infty$$

$$\text{(B) } \textit{Slow Time-Scale:} \quad \sum_{t=0}^{\infty} \gamma(t) \to \infty, \qquad \sum_{t=0}^{\infty} \gamma^2(t) < \infty$$

$$\text{(C)} \quad \lim_{t\to\infty} \frac{\gamma(t)}{\alpha(t)} \to 0$$

An example of possible step sizes for which this condition is true is $\alpha(t) = \frac{1}{t}$ and $\gamma(t) = \frac{1}{t \log t}$. In our experiments we use $\alpha(t) = \frac{C}{\lceil \frac{t}{D} \rceil}$, and $\gamma(t) = \frac{C'}{1+\lceil \frac{t \log(t)}{D} \rceil}$.

We then detail the proof for Theorem 4.3 below. This proof involves mapping the discrete Q and λ updates from the MAIQL algorithm (Section 4) to updates in an equivalent continuous-time Ordinary Differential Equation (ODE). This conversion then allows us to use the analysis tools created to analyse the evolution of two-timescale ODEs to show that our coupled updates converge. The proof detailed below broadly follows along the lines of Avrachenkov and Borkar [3], but where they discuss convergence in the binary action case, we generalize their proof to the multi-action scenario by using the notion of multi-action indexability from [12].

Theorem 4.3. *MAIQL converges to the optimal multi-action index $\lambda^*_{s,a}$ for a given state s and action a under Assumptions 1, 2, 3, and the problem being multi-action indexable.*

Proof. To convert these discrete updates to ODEs, we map a given time-step t to a point $\tau = T(t)$ in a continuous time, such that any time $T(t) = \sum_{m=0}^{t} \alpha(t)$. Because we're parameterising the time with α (rather than γ) we call τ the fast time-scale. To make this more concrete, we define $Q(\tau)$ as a function of the Q-value with time, and set $Q(T(t)) = Q^t$ to the value of the Q-function after t updates . Then, for values of $T(t) < \tau < T(t+1)$, $Q(\tau)$ is assumed to be linearly interpolated between Q^t and Q^{t+1}, creating a continuous function of τ. Similarly, we define $\lambda(\tau)$ such that $\lambda(T(t)) = \lambda^t$

We can then re-arrange the terms in Equation 9 to create an ODE that characterises the value of $Q(\tau)$:

$$Q^{t+1}(s,a_j) = Q^t(s,a) + \alpha(t)\big[[r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j \in \{0,1\}} Q^t(s',a'_j)] - Q^t(s,a_j)\big]$$

$$\Rightarrow \underbrace{\frac{Q^{t+1}(s,a_j) - Q^t(s,a_j)}{\alpha(t)}}_{\dot{Q}(\tau)} = [r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j \in \{0,1\}} Q^t(s',a'_j)] - Q^t(s,a_j)$$

where $\dot{Q}(\tau)$ is the derivative of $Q(\tau)$ and corresponds to the slope of the interpolated function in the range $(T(t), T(t+1))$.
Similarly, we can re-arrange Equation 10 to get the ODE for $\lambda(\tau)$:

$$\lambda^{t+1}_{s,a_j} = \lambda^t_{s,a_j} + \alpha(t)\left(\frac{\gamma(t)}{\alpha(t)}\right)(Q^t(s,a_j) - Q^t(s,a_{j-1}))$$

$$\Rightarrow \underbrace{\frac{\lambda^{t+1}_{s,a_j} - \lambda^t_{s,a_j}}{\alpha(t)}}_{\dot{\lambda}(\tau)} = \left(\frac{\gamma(t)}{\alpha(t)}\right)(Q^t(s,a_j) - Q^t(s,a_{j-1})) \tag{16}$$

Then, if look at Equation 16, we see $\lim_{\tau\to\infty} \dot{\lambda}(\tau) \to 0$ because, by Assumption 3 (c), $\lim_{t\to\infty} \frac{\gamma(t)}{\alpha(t)} \to 0$ and, by Assumption 3 (A), $T(\infty) = \sum_{t=0}^{\infty} \alpha(t) \to \infty$. Therefore, $\lambda(\tau)$ can be seen as quasi-static w.r.t. $Q(\tau)$ at the fast time-scale. As a result, the updates in this time-scale correspond to standard Q-Learning for a fixed MDP defined by the value of $\lambda(\tau)$. Given Assumptions 1, 3 (A), and 2, this is known to converge to the optimal Q-values Q^*_λ for the given value of $\lambda(\tau)$ [1].

Now, at the slow time-scale τ', we can repeat this continuous-time re-parameterisation, except with $T'(t) = \sum_{m=0}^{t} \gamma(t)$. Then, re-arranging Equation 9 in a similar way as above, we get:

$$\underbrace{\frac{Q^{t+1}(s,a_j) - Q^t(s,a_j)}{\gamma(t)}}_{\dot{Q}(\tau')} = \left(\frac{\alpha(t)}{\gamma(t)}\right)[r(s) - \lambda^t_{s,a_j} c_j - f(Q^t) + \max_{a'_j \in \{0,1\}} Q^t(s',a'_j)] - Q^t(s,a_j)$$

Now, given that $\lim_{t\to\infty} \frac{\alpha(t)}{\gamma(t)} \to \infty$, and from the argument above about the Q-values converging in the fast time-scale, we can see the interpolated $\lambda(\tau')$ value as tracking the converged Q-values $Q^*_{\lambda(\tau')}$ (for that value of $\lambda(\tau')$). Then, we can write the ODE for $\lambda(\tau')$ as:

$$\dot{\lambda}(\tau') = Q^*_{\lambda(\tau')}(s,a_j) - Q^*_{\lambda(\tau')}(s,a_{j-1})$$

where $Q^*_{\lambda(\tau')}$ corresponds to the optimal Q-values corresponding to the given value of $\lambda(\tau')$.

Now, if $\lambda(\tau') < \lambda^*_{s,a_j}$ (the multi-action index for state s and action a_j), by the definition of the multi-action index from the main text, we know that an action of weight c_j or higher is preferred. As a result, we see that $\dot{\lambda}(\tau') > 0$ in that case. If $\lambda(\tau') > \lambda^*_{s,a_j}$, the opposite is true and so $\dot{\lambda}(\tau') < 0$. Then, because $\lambda(0) = 0$ is bounded and given the step-sizes in Assumption 3 (B), $\lambda(\tau')$ converges to an equilibrium in which $Q^*_\lambda(s, a_j) - Q^*_\lambda(s, a) \to 0$.

Given that, by definition, $\lambda^*(s, a_j)$ is the value at which $Q^*_\lambda(s, a_j) = Q^*_\lambda(s, a_{j-1})$, $\lambda(\tau')$ converges to the multi-action index. □

This is a high-level proof, but the specific conditions for convergence can be seen in Lakshminarayanan and Bhatnagar [20]. They require 5 conditions: (1) Lipschitzness, (2) Bounded 'noise', (3) Properties about the relative step-sizes, (4) Convergence of fast time-scale, and (5) Convergence of slow time-scale.

Of these, (1)-(4) proceed in much the same way as in Avrachenkov and Borkar [3] because they do not depend on the multi-action extension of indexability. In addition, it is easy to show that the proof of (5) from Avrachenkov and Borkar [3] extends to the multi-action case which considers the limiting value of $Q(s, a_j) - Q(s, a_{j-1})$ rather than $Q(s, 1) - Q(s, 0)$. As a result, we refer the reader to Avrachenkov and Borkar [3] for the complete proof.

B REPRODUCIBILITY

Code is available at https://github.com/killian-34/MAIQL_and_LPQL. All the Q and λ values are initiated to zero in all the experiments. The parameter settings used for the two process type, random, and medication adherence data experiments are included in Tables 1, 2, and 3, respectively. C is the multiplier for the size of the Q-value updates. C' is the multiplier for the size of the index value updates. "Rp/dream" is the number of replays per dream. "Rp T" is the replay period (replay every T steps). λ-bound is the upper bound (and negative of the lower bound) imposed on values of the indices for WIBQL and MAIQL during learning – placing these bounds sometimes helps prevent divergent behavior in early rounds when updates are large – $\lambda_{\max}$ is the upper bound value that an index could take, as defined by the problem parameters, i.e., $\frac{\max\{r\}}{\min\{C\}(1-\beta)}$ [18]. D is the divisor of the decaying ϵ-greedy function as well as the divisor of $\alpha(t)$ and $\gamma(t)$, the decaying functions defining the size of the updates of Q-values and index values, defined in the previous section. ϵ_0 is the multiplier for the ϵ-greedy function. n_{lam} is the number of points in λ-space used to approximate the Q(s, a, λ)-functions in LQPL and MAIQL-Aprx. All values were determined via manual tuning – empirically we found that most parameter settings led to similar long-term performance between algorithms, as long as the settings did not cause the algorithms to diverge. In the tables, M-Aprx stands for MAIQL-Aprx.

C ALGORITHM PSEUDOCODES

See Algorithms 1 and 2 for the update and action selection steps of MAIQL and Algorithms 3 and 4 for the update and action selection steps of LPQL. The linear program for Oracle-LP-Index for a given current state s_{cur} and action a_k is given below:

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	0.1	0.2	0.1	0.4	0.4
C'	0.2	-	0.2	-	-
Rp/dream	NA	1000	1000	1000	NA
Rp T	1E+06	100	10	100	1E+06
λ-bound	3	-	3	3	3
D	500	500	500	500	500
ϵ_0	0.99	0.99	0.99	0.99	0.99
n_{lam}	-	-	-	3000	3000

Table 1: Parameter settings for two process experiment.

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	-	-	0.2	0.8	0.8
C'	-	-	0.4	-	-
Rp/dream	-	-	1000	NA	NA
Rp T	-	-	100	1E+06	1E+06
λ-bound	-	-	$\lambda_{\max}$	$\lambda_{\max}$	$\lambda_{\max}$
D	-	-	500	500	500
ϵ_0	-	-	0.99	0.99	0.99
n_{lam}	-	-	-	2000	2000

Table 2: Parameter settings for random data experiment.

	WIBQL	QL-λ=0	MAIQL	M-Aprx	LPQL
C	-	0.8	0.05	0.8	0.8
C'	-	-	0.1	-	-
Rp/dream	-	1000	1000	1000	1000
Rp T	-	10	5	5	5
λ-bound	-	-	$\lambda_{\max}$	$\lambda_{\max}$	$\lambda_{\max}$
D	-	1000	2000	1000	1000
ϵ_0	-	0.99	0.99	0.99	0.99
n_{lam}	-	-	-	2000	2000

Table 3: Parameter settings for adherence data experiment.

$$
\begin{aligned}
\min_{V^i(s^i,\lambda^i),\lambda^i} & \sum_{i=0}^{N-1} \frac{\lambda^i B}{1-\beta} + \sum_{i=0}^{N-1} \mu^i(s^i) V^i(s^i, \lambda^i) \\
\text{s.t. } & V^i(s^i, \lambda^i) \geq r^i(s^i) - \lambda^i c_j + \beta \sum_{s^{i\prime}} T(s^i, a^i_j, s^{i\prime}) V^i(s^{i\prime}, \lambda^i) \\
& \qquad \forall i \in \{0, ..., N-1\}, \quad \forall s^i \in \mathcal{S}, \quad \forall a_j \in \mathcal{A} \\
& r^i(s^i_{cur}) - \lambda^i c_k + \beta \sum_{s^{i\prime}} T(s^i_{cur}, a^i_k, s^{i\prime}) V^i(s^{i\prime}, \lambda^i) = \\
& \qquad r^i(s^i_{cur}) - \lambda^i c_{k-1} + \beta \sum_{s^{i\prime}} T(s^i_{cur}, a^i_{k-1}, s^{i\prime}) V^i(s^{i\prime}, \lambda^i) \\
& \qquad \forall i \in \{0, ..., N-1\} \\
& \lambda^i \geq 0 \quad \forall i \in \{0, ..., N-1\}
\end{aligned}
\tag{17}
$$

The LP is similar to Eq. 11, but differs in two ways. First, instead of having a single λ value across all arms, each arm has its own independent λ^i value. Second, the second group of constraints is new, and forces the λ^i values to be set such that the planner would be indifferent between taking the action in question a_k or the action that is one step cheaper a_{k-1}, which follows exactly the definition of the multi-action indexes. Note that although the indexes can each be computed independently, for convenience, we compute the index for a given a_k for each arm simultaneously to reduce overhead, as given in the above LP.

The ActionKnapsackILP referenced in Algorithm 4 is the same as the modified knapsack given in Killian et al. [18], reproduced below:

$$\begin{aligned}
&\max_{\boldsymbol{A}} \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} \boldsymbol{A}_{ij} Q_{\boldsymbol{s},\lambda_{ind}}(i, a_j) \\
&\text{s.t.} \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} \boldsymbol{A}_{ij} c_j \le B \\
&\sum_{j=0}^{M-1} \boldsymbol{A}_{ij} = 1 \quad \forall i \in 0,\dots,N-1 \\
&\boldsymbol{A}_{ij} \in \{0,1\}
\end{aligned} \tag{18}$$

where $Q_{\boldsymbol{s},\lambda_{ind}}(i, a_j)$ is the Q-function for each arm filtered to the current state of the arms, $\boldsymbol{s}$, and minimizing value λ_{ind}, as given by the penultimate line of Algorithm 4.

RandomAction, referenced in Algorithms 2 and 4, chooses random actions through the following iterative procedure: (1) randomly choose an arm with uniform probability, (2) randomly choose an action with probability inversely proportional to one plus its cost (must add one to avoid dividing by 0 for no-action). The procedure iterates until the budget is exhausted.

Algorithm 1: MAIQL Update

```
Data: $Q \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)\times|\mathcal{S}|\times|\mathcal{A}|}$,   // Need one copy
      of $Q[s,a]$ for each index on each arm
  $\lambda \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)}$,            // multi-action index
      estimates
  Batch, $C$,          // Experience tuples, action costs
  $t, \nu(\cdot)$,           // iteration, state-action counter
  $\mathcal{S}, \mathcal{A}, N$   // state space, action space, # of arms
Hyperparameters: $\beta, C, C', D$              // See maintext
for $(n, s, a, r, s') \in Batch$ do
    $\alpha = \frac{C}{\lceil \frac{\nu(s,a,n)}{D} \rceil}$
    for $i \in 0, \dots, |\mathcal{S}|$ do
        for $j \in 1, \dots, |\mathcal{A}|$ do
            $Q[n,i,j,s,a] \mathrel{+}= \alpha(r - C[a]*\lambda[i,j] + \beta *$
              $\max\{Q[n,i,j,s']\} - Q[n,i,j,s,a])$
    if $a \neq 0$ & $t \pmod N == 0$ then
        $\gamma = \frac{C'}{1+\lceil \frac{\nu(s,a,n)\log \nu(s,a,n)}{D} \rceil}$
        $\lambda[s,a] \mathrel{+}= \frac{\gamma(Q[n,s,a,s,a]) - Q[n,s,a,s,a-1])}{C[a]-C[a-1]}$
return $Q, \lambda$
```

Algorithm 2: MAIQL Action Select

```
Data: $\lambda \in \mathbb{R}^{N\times|\mathcal{S}|\times(|\mathcal{A}|-1)}$,      // multi-action index
      estimates
  $\boldsymbol{s} \in \mathbb{R}^N$              // current state of all arms
  $t, N, B$   // current iteration, # of arms, budget
if EpsilonGreedy$(t)$ then
    return RandomAction()
else
    $\boldsymbol{a} = [0$ for _ in range$(N)]$
    $\lambda_f =$ FilterCurrentState$(\lambda, \boldsymbol{s})$  // $\lambda_f \in \mathbb{R}^{N\times(|\mathcal{A}|-1)}$
    for $i \in 0 \dots B$ do
        $i = \arg\max(\lambda_f[\boldsymbol{a}+1] - \lambda_f[\boldsymbol{a}])$ // $\boldsymbol{a}$ is a vector
          index, arg max ignores out of bounds
          indexes
        $a[i] \mathrel{+}= 1$
return $\boldsymbol{a}$
```

Algorithm 3: LPQL Update

```
Data: $Q \in \mathbb{R}^{N\times n_{lam}|\mathcal{S}|\times|\mathcal{A}|}$,        // Need one copy of
      $Q[s,a]$ for each of the $n_{lam}$ test points on
      each arm
  Batch, $C$,       // Experience tuples, action costs
  $\lambda_{\max}$,           // Max $\lambda$ at which to estimate $Q$
  $n_{lam}$,    // # of $\lambda$ points at which to estimate $Q$
  $\nu(\cdot)$                      // state-action counter
Hyperparameters: $\beta, C, D$             // See maintext
for $(n, s, a, r, s') \in Batch$ do
    $\alpha = \frac{C}{\lceil \frac{\nu(s,a,n)}{D} \rceil}$
    for $i \in 0, \dots, n_{lam}$ do
        $\lambda_p = \frac{i*\lambda_{\max}}{n_{lam}}$
        $Q[n,i,s,a] \mathrel{+}=$
          $\alpha(r - C[a]*\lambda_p + \beta*\max\{Q[n,i,s']\} - Q[n,i,s,a])$
return Q
```

EpsilonGreedy(t), also referenced in Algorithms 2 and 4, draws a uniform random number between 0 and 1 and returns true if it is less than $\epsilon_0/\lceil \frac{t}{D} \rceil$ and false otherwise.

D MEDICATION ADHERENCE SETTING DETAILS

We used the following procedure to estimate transition probabilities from the medication adherence data from Killian et al. [19]. First, we specify a history length of L. This gives a state space of size 2^L for each arm. Then, for each patient in the data, we count all of the occurrences of each state transition across a treatment regimen of 6 months (168 days). If L was small (e.g., 1 or 2), we could take a frequentist approach and simply normalize these counts appropriately to get valid transition probabilities to sample for experiments. However, as the history length L gets larger, the number of nonzero entries in the count data for state transitions become large. We take two steps to account for this sparsity. (1) We run K-means clustering over all patients, using the count data as features, then

Algorithm 4: LPQL Action Select

```
Data: $Q \in \mathbb{R}^{N \times n_{lam} |\mathcal{S}| \times |\mathcal{A}|}$,  // $Q$-functions for each
        of the $n_{lam}$ test points on each arm
  $s \in \mathbb{R}^N$                // current state of all arms
  $\lambda_{\max}$,            // Max $\lambda$ at which $Q$ is estimated
  $n_{lam}$,   // # of $\lambda$ points at which $Q$ is estimated
  $t, \beta$                  // iteration, discount factor
  $N, C, B$          // # of arms, action costs, budget
if EpsilonGreedy($t$) then
    return RandomAction()
$a$ = [0 for _ in range ($N$)]
$Q_f$ = FilterCurrentState($Q, s$)   // $Q_f \in \mathbb{R}^{N \times n_{lam} \times |\mathcal{A}|}$
$\lambda_{ind} = -1$
/* The min of Eq. 11 occurs at the point where
   the negative sum of slopes of all $V^i = \max\{Q^i_\lambda\}$
   is $\leq B/(1-\beta)$, so we will iterate through our
   estimates of $Q^i_\lambda$ and stop our search at the
   first point where that is true. */
for $i \in 0, \ldots, n_{lam}$ do
    $\lambda^0_p = \frac{i * \lambda_{\max}}{n_{lam}}$
    $\lambda^1_p = \frac{(i+1) * \lambda_{\max}}{n_{lam}}$
    $m_V = \frac{\max_a\{Q_f[:, i+1]\} - \max_a\{Q_f[:, i]\}}{\lambda^1_p - \lambda^0_p}$        // $m_V \in \mathbb{R}^N$
    if $\sum_n \{m_V\} \geq -\frac{B}{1-\beta}$ then
        $\lambda_{ind} = i$
        break
$a$ = ActionKnapsackILP($Q_f[:, \lambda_{ind}, :], C, B$)
return $a$
```

combine the counts for all patients within a cluster. Intuitively, the larger the K, the more "peaks" of the distribution of patient adherence modes we will try to approximate, but the fewer data points are available to estimate the distribution in each cluster — however, it may be desirable to have more clusters to allow for some samples to come from uncommon but "diverse" modes that may be challenging to plan for. In this paper, we set K to 10. (2) We then take a Bayesian approach, rather than a frequentist approach for sampling patients/processes from the clustered counts data. That is, we treat the counts as priors of a beta distribution, then sample transition probabilities from those distributions according to the priors. Finally, to simulate action effects, since actions were not recorded in the available adherence data, we scale the priors multiplicatively according to the index of the action, i.e., larger/more expensive actions increase the priors associated with moving to the adhering state.

In summary, to get a transition function for a single simulated arm in the medication adherence experimental setting, we do the following. First, randomly choose a cluster, with probability weighted by the number of patients in the cluster. Then, build up a transition matrix by sampling each row according to its own beta distribution with priors given by the counts data (i.e., actual observations of $s \rightarrow s'$ transitions), scaled by the action effects.

This process was desirable for producing simulated arms with transition functions tailored to resemble that of a real world dataset, while allowing for some randomness via the sampling procedure, as well as a straightforward way to impose simulated action effects. However, one downside of this approach is that, since each row of the transition matrix is sampled independently, this may produce simulated arms whose probability of adherence changes in a non-smooth manner as a function of history. For example, in the real-world, we would expect that $P(0111 \rightarrow 1111)$ is correlated with $P(1011 \rightarrow 0111)$ and that $P(0000 \rightarrow 0000)$ is correlated with $P(1000 \rightarrow 0000)$, but our procedure would not necessarily enforce these relationships if there were not sufficient occurrences of each transition in the counts data.

The python code used to execute this procedure is included in the repository at https://github.com/killian-34/MAIQL_and_LPQL.