COMS E6998.001: Bandits and Reinforcement Learning **Fall 2017**

Reinforcement Learning

Lecturer: Alekh Agarwal *Lecture 10, November 8*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

In this lecture, we will introduce reinforcement learning. Reinforcement learning (RL) is a strict generalization of the contextual bandit (CB) problem. Like CBs, it consists of a sequential interaction between the learning agent and its environment. Unlike CBs, the agents actions have long-term consequences now. That is, the actions influence now only the reward acquired at the current time-step, but also influence the distributions over future rewards which an agent might acquire. We will first present the basic setup of the general RL problem now, before discussing the questions of interest and solutions to them.

10.1 Problem Setup

At each round $t = 1, 2, \ldots,$:

1. Environment reveals the current state $s_t \in \mathcal{S}$.
2. Learner chooses an action $a_t \in \mathcal{A}$.
3. Environment reveals a reward $r_t \in [0, 1]$ and draws the next state s_{t+1} based on a_t.

Figure 10.1: The repeated interaction between the environment and a learning agent in the reinforcement learning problem

The basic interaction between a learning agent and its environment is described in Figure 10.1. As we can see, this bears some similarities and differences to the corresponding interaction in CBs (Figure 6.1). One key difference is the notion of a *state*. States can be thought of a bit like contexts in CBs. One important difference is that when the agent takes an action a_t, it influences not just the current reward, but also the future state s_{t+1}. Consequently, a poor action at the current time-step can not only result in a low reward r_t, but also place us in a future state s_{t+1} from which no significant rewards are possible in thereafter! At the first glance, this dependence of the future state on the agent's actions might appear similar to adversarial CBs, where the adversary can choose future contexts based on the current action. However, we will next introduce the learning setting formally, whence the distinctions will be clearer.

Definition 10.1 (Episodic Markov Decision Process (MDP)) *An Episodic Markov Decision Process (MDP) M is parametrized by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \Gamma, \nu, H)$ where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ denotes the action space. The initial state is drawn as $s_1 \sim \mu$, while the corresponding states obey $s_{t+1} \sim \Gamma(s|s_t, a_t)$. The instataneous rewards are drawn according to $r_t \sim \mathcal{R}(s_t, a_t)$. The entire loop in Figure 10.1 repeats for H steps, for a fixed horizon H after which the process ends.*

Remark: The decision process is called *Markovian* as the reward r_t only depends on s_t and a_t, as does the future state s_{t+1}. Consequently, given the current state s_t of an agent, the prior actions $a_1, \ldots, a_{t-1}$ which it took to reach s_t do not matter to the future. This is distinct from non-Markovian processes where the entire path followed to s_t might matter in the future.

The learning goal in an MDP consists of maximizing the cumulative reward. Given a policy $\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$, which maps states to distributions over actions, we define the value of π as

$$V(\pi, M) = \mathbb{E}_{(s_1,a_1,r_1,\ldots,s_H,a_H,r_H)\sim M,\pi}[r_1 + \ldots + r_H], \tag{10.1}$$

where the notation $(s_1, a_1, r_1, \ldots, s_H, a_H, r_H) \sim M, \pi$ refers to the states $s_1 \sim \nu$, $s_{t+1} \sim \Gamma(s|s_t, a_t)$, $r_t \sim \mathcal{R}(s_t, a_t)$ and $a_t \sim \pi(s_t)$ for all $t = 1, 2, \ldots, H$. That is, the notation referes to all the actions chosen according to π, while the remaining quantities are drawn based on the MDP M given these actions. Often, the policies π we consider are *non-stationary*, that is, the action a_t depends both on s_t and the time-step t. In such a case, we formally think of a non-stationary policy π as a sequence of mappings $\pi_1, \ldots, \pi_H$ with $a_t \sim \pi_t(s_t)$.

The learning goal of the agent is to find a non-stationary policy π which maximizes the value, that is, $\pi^\star_M = \arg\max V(\pi, M)$. When the MDP M in consideration is clear from the context, we will suppress the argument M for conciseness.

Example 1 (Navigation) *Navigation is perhaps the simplest to see example of RL. The state of the agent is their current location. The four actions might be moving 1 step along each of east, west, north or south. The transitions in the simplest setting are deterministic. Taking the north action moves the agent one step north of their location, assuming that the size of a step is standardized. The agent might have a goal state g they are trying to reach, and the reward is 0 until the agent reaches the goal, and 1 upon reaching the goal state. Since we are in an episodic setting with length H episodes, we also need to define what happens if the agent reaches a goal at a step $t < H$, and we can imagine that they transition to a* no-op *state s_0 from g with probability 1 irrespective of the action, which does not accrue any further rewards. Alternatively, there can be a self-loop at g which means the agent keeps getting a reward of 1 every time step they take this self-loop. This second formulation would mean that the agent gets a higher reward by getting to g faster.*

In the above versions, the agent gets no intermediate feedback on the quality of their actions till they get to the goal state. A more helpful setting might be one of reward shaping*, where the agent gets a reward after every action based on their distance from the goal state. Note however, that we have to exercise care while doing this in an episodic setting. Suppose we say simply that the reward is equal to negative of the number of steps from the goal, there is a large positive reward for getting to the goal and the agent transitions to a state s_0 from g getting no future reward using a stop action, or can move to any of the neighboring locations using the standard movement actions. Then an optimal agent quickly gets to the goal, then takes a step away from the goal in any direction getting a small negative reward, followed be returning to g to get the positive reward again and so on. The lesson is that designing a numerical reward function such that the optimal policy under it has the intuitively desirable behavior can be quite non-trivial in practice.*

Navigation is a good example in that it is clearly beyond CB. The agent's next location is integrally dependent on the previous one, and getting to a target location which yields rewards cannot be achieved by a single action, but requires a carefully planned sequence of actions.

Example 2 (Conversational agent) *This is another fairly natural RL problem. The state of an agent can be the current transcript of the conversation so far, along with any additional information about the world, such as the context for the conversation, characteristics of the other agents or humans in the conversation etc. Actions depend on the domain. In the rawest form, we can think of it as the next statement to make in the conversation. Sometimes, conversational agents are designed for task completion, such as travel assistant or tech support or a virtual office*

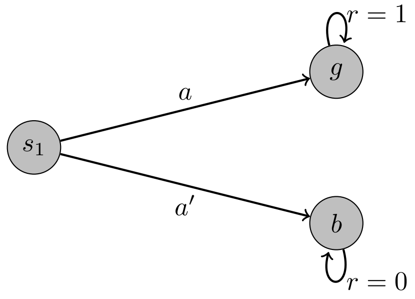

Figure 10.2: An example illustrating difference between RL and CBs. There are 3 states. Action a in state s_1 transitions to state g with probability 1, while a' in s_1 transitions to b. There is only one available action in each of g and b which just loops back to the same state, with a reward of 1 at each step for g and 0 for b.

receptionist. In these cases, there might be a predefined set of slots *which the agent needs to fill before they can find a good solution. For instance, in the travel agent case, these might correspond to the dates, source, destination and mode of travel. The actions might correspond to natural language queries to fill these slots.*

In task completion settings, reward is naturally defined as a binary outcome on whether the task was completed or not, such as whether the travel was successfully booked or not. Depending on the domain, we could further refine it based on the quality or the price of the travel package found. In more generic conversational settings, the ultimate reward is whether the conversation was satisfactory to the other agents or humans, or not.

Example 3 (Board games) *This is perhaps the most popular category of RL applications, where RL has been successfully applied to solve Backgammon, Go and various forms of Poker. For board games, the usual setting consists of the state being the current game board, actions being the potential next moves and reward being the eventual win/loss outcome or a more detailed score when it is defined in the game.*

RL versus Contextual Bandits: We can now more crisply draw the distinction between RL and CB. In RL, we measure each policy based on $V_M(\pi)$. Crucially, the states $s_2, \ldots, s_H$ have different distributions depending on the actions chosen by the policy π. In CB, we evaluate each policy π under the *same distribution on contexts and rewards*. Even when this distribution is adversarially generated in response to the algorithm, we then freeze these adversarial contexts and rewards and evaluate every policy for this sequence. As an extreme example, consider the example MDP shown in Figure 10.2. Here we have just 3 states, with 2 actions in the first state and just one action in the remaining states. There is no reward received for either choice of the action a_1. If the agent chooses the action leading to the good state g, then the reward is 1 for the remaining $H - 1$ steps, and 0 otherwise. In this case, a policy which transitions to the bad state b on the first step has a value of 0, while a policy transitioning to g has a value of $H - 1$. In a contextual bandit setting though, if the algorithm were to act in the same environment, and transition to the state b on the first step, then it obtains a cumulative reward of 0. But any other policy evaluated on this distribution of rewards also gets a reward of only 0, meaning that the algorithm would be considered optimal under the contextual bandit notion of regret!

Finite versus infinite horizons: The way we have defined an MDP is *episodic*, that is the interaction between the learner and environment proceeds in episodes of a finite length H. An alternative definition of MDPs consists of the infinite horizon setting, which can be thought of as Definition 10.1 with $H = \infty$. Since the cumulative reward can be infinite in such settings, people typically consider a discounted reward, where a discount factor $\gamma \in (0, 1)$ is used to trade-off the reward in the current time-step with what we might obtain in the future. We will not discuss this discounted, infinite horizon setting in the lecture, and refer the students to a standard text such as Puterman [1994] for the corresponding definitions.

States versus contexts: States appear to play a similar role in RL as contexts do in CB. That is, the agent perceives the state and takes actions based on it. However, the notion of a state in typical RL literature carries some more meaning to it, in that states are typically viewed as *sufficient statistics* to summarize the entire preceding trajectory. Thus the context in CB is usually a modeling choice. We can choose to put more information or different features in a context, which leads to a different type of policy. States in RL are often a property of the problem, and we base our policies on states since they contain all the information required to choose the right action. Finally, we think of contexts as high-dimensional vectors in CB. Often in RL, and definitely in this lecture, we will think of states as discrete and small in number. That is, $\mathcal{S}$ will be assumed to be a set of small cardinality. This will also be reflected in our desire to find the best policy from all possible mappings of states to actions, rather than worrying about restricted policy classes which can be used to generalize across contexts. When the policy is maintained as a table prescribing an action for each state explicitly, the representation is called *tabular*.

10.2 Value functions

A central notion in reinforcement learning is that of value functions. Let us consider a non-stationary policy π (that is, a sequence of policies $\pi_1, \pi_2, \ldots, \pi_H$). Then we define the value functions:

$$\begin{aligned} V_t^\pi(s, M) &= \mathbb{E}_{(s_t,a_t,r_t,\ldots,s_H,a_H,r_H)\sim M,\pi}[\sum_{\tau=t}^{H} r_\tau | s_t = s], \quad \text{and} \\ Q_t^\pi(s, a, M) &= \mathbb{E}_{(s_t,a_t,r_t,\ldots,s_H,a_H,r_H)\sim M,\pi}[\sum_{\tau=t}^{H} r_\tau | s_t = s, a_t = a]. \end{aligned} \tag{10.2}$$

In words, $V_t^\pi(s, M)$ refers to the expected cumulative reward that the agent can obtain by starting in state s at time step t and following the policy π for the subsequent steps. Clearly our previous definition $V(\pi, M)$ coincides with $\mathbb{E}_{s\sim\nu}[V_1^\pi(s)]$, with the MDP M being implicit. Similarly, the $Q_t^\pi(s, a, M)$ function captures the expected cumulative reward obtained by taking a in state s at time t, and following π thereafter. Once again, we will generally suppress the dependence on M when it is clear from the context.

One important property of these value functions is that they can be recursively defined. Indeed, based on our definitions, it is easily checked that we have for any non-stationary policy π and any time step $t = 1, 2, \ldots, H$,

$$\begin{aligned} V_t^\pi(s, M) &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim M,\pi}[r_t + V_{t+1}^\pi(s_{t+1}, M) | s_t = s] \\ &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},a_{t+1})\sim M,\pi}[r_t + Q_{t+1}^\pi(s_{t+1}, a_{t+1}, M) | s_t = s] \end{aligned} \tag{10.3}$$

$$\begin{aligned} Q_t^\pi(s, a, M) &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim M,\pi}[r_t + V_{t+1}^\pi(s_{t+1}, M) | s_t = s, a_t = a] \\ &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1},a_{t+1})\sim M,\pi}[r_t + Q_{t+1}^\pi(s_{t+1}, a_{t+1}, M) | s_t = s, a_t = a], \end{aligned} \tag{10.4}$$

where we define $V_{H+1}^\pi(s) = Q_{H+1}^\pi(s, a) = 0$ for all s, a, π. The correspondence of this definition with the earlier ones (10.2) can be seen inductively. The two clearly coincide when $t = H$, since we just count the expected instantaneous reward. Assuming that the two definitions agree for all $t = H, \ldots, \tau$, showing that they hold for $\tau - 1$ is easy as well as we are just unfolding the sum.

The recursive definitions are handy because they suggest an obvious algorithm to compute V_t^π and Q_t^π, if we are given an MDP M and a policy π. We can simply use dynamic programming where we start from the easily computed $t = H$, and then *back-up* these values one level by adding in the rewards at the previous step.

So far, we have defined the notions of value, given a policy π. Of course, our goal is to find the best possible policy. As mentioned previously, we will consider all possible mappings from states to actions as policies. Formally, our goal is to find $\pi^\star = (\pi_1^\star, \ldots, \pi_H^\star)$ which maximizes $V(\pi, M)$. The V and Q functions associated with $\pi^\star$ are referred to as $V^\star$ and $Q^\star$ respectively. That is, we use $V_t^\star(s, M)$ to denote the largest possible cumulative reward in the MDP M, when we are in state s at time t. Similarly, $Q_t^\star(s, a, M)$ is the largest cumulative reward in M after taking action a in state s and acting optimally thereafter. We now present recursive formulations for $V^\star$ and $Q^\star$, analogous to our earlier definitions of V^π and Q^π.

Theorem 10.2 (Bellman Optimality Equations) *Let $V_t^\star$ and $Q_t^\star$ be the value functions corresponding to an optimal policy $\pi^\star$. Then we have the following relations*

$$V_t^\star(s, M) = \max_{a \in \mathcal{A}} \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim M}[r_t + V_{t+1}^\star(s_{t+1}, M) | s_t = s, a_t = a]$$
$$Q_t^\star(s, a, M) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}) \sim M}[r_t + V_{t+1}^\star(s_{t+1}, M) | s_t = s, a_t = a]$$

Proof: The proof follows by induction, along with the fact that our policies have a tabular structure. We present the proof for $V^\star$ here, with the argument being extremely similar for $Q^\star$ as well. Starting with the base case at level H, we define $V_{H+1}^\star \equiv 0$ for all states as before. $V_H^\star(s, M)$ is defined as the expected reward achieved by following the optimal policy at step H in state s. Since each policy has a tabular structure, we can choose the best action for state s in step H, independently of the choices we make at other steps t or states s'. This implies that

$$V_H^\star(s, M) = \max_{a \in \mathcal{A}} \mathbb{E}_{r_H \sim M}[r_H | s_H = s, a_H = a],$$

which coincides with the assertion of the lemma at $t = H$. Now we assume that the lemma holds for $s = H, H - 1, \ldots, t + 1$, and establish it at $s = t$. By definition, we have

$$\begin{aligned}
V_t^\star(s, M) &= \max_{a_t, \ldots, a_H} \mathbb{E}_{(s_t, a_t', r_t, \ldots, s_H, a_H', r_H) \sim M} \left[\sum_{s=t}^{H} r_s | s_t = s, a_s' = a_s : s = t, \ldots, H \right] \\
&= \max_{a_t} \mathbb{E}_{(s_t, a_t', r_t) \sim M} \left[r_t + \max_{a_{t+1}, \ldots, a_H} \mathbb{E}_{(s_{t+1}, a_{t+1}', r_{t+1}, \ldots, s_H, a_H', r_H)} \left[\sum_{s=t+1}^{H} r_s | a_s' = a_s : s = t+1, \ldots, H \right] | s_t = s, a_t' = a_t \right] \\
&= \max_{a_t} \mathbb{E}_{(s_t, a_t', r_t) \sim M} \left[r_t + \mathbb{E}_{s'}[V_{t+1}^\star(s', M)] | s_t = s, a_t' = a_t \right].
\end{aligned}$$

Here the second equality uses the tabular structure of the policies strongly, by saying that having made optimal choices for times $t+1$ through H to obtain value $V_{t+1}^\star$, we just need to pick the action a_t greedily according to the instantaneous reward and the best value from the next state. ■

Remark: The argument here is very dynamic programming like, and might be reminiscent of shortest path like algorithms. Indeed, the recursion here is a generalization of the argument used in shortest path algorithms. The shortest path from a point x to y is given by moving from x to the neighbor z which is closest to y.

From value functions to optimal policy: So far, we have seen how to compute the optimal value functions in an MDP, but we would also like to know the optimal policy in order to take actions in the MDP. Given $Q^\star$ or $V^\star$, the optimal policy is simply determined as

$$\pi_t^\star(s) = \max_{a \in \mathcal{A}} Q_t^\star(s, a) = \max_{a \in \mathcal{A}} \mathbb{E}[r(a) + V^\star(s_{t+1}) | s_t = s]. \tag{10.5}$$

These equalities can again be seen to be inductively true. At step H, $Q^\star(s, a)$ is simply the reward obtained by taking action a in state s and $\pi^\star$ simply picks the action with the largest immediate reward at the final step given the current state, by the Markovian structure of the process and the tabular formulation of the policy class. Assuming that the statement holds at all steps $s = t + 1, \ldots, H$, it also holds at step t by simply utilizing the recursive definition of $Q^\star$ in Theorem 10.2. Finally, the expression in terms of $V^\star$ follows from the definition of $Q^\star$ in terms of $V^\star$.

10.3 Learning optimal behavior from samples

So far, we have seen how to compute the optimal policy and value functions given that we know everything about the MDP, that is the reward and transition functions are known. But in practice, our learning agent can only act in the MDP and experience the reward and transitions through states and rewards it perceives in response to its actions. And we would like to learn an optimal behavior policy in an MDP, once the agent has experienced a large enough number of trajectories.

As a natural starting point, we might attempt to treat RL as a more advanced type of a bandit problem, and try applying existing bandit algorithms to find a good RL policy. The most natural idea is to take a CB algorithm and apply it with the state as context in RL settings. However, as we discussed before, the notion of regret in CB is not strong enough to imply meaningful guarantees in an RL setting.

We next present an algorithm for episodic RL based on the elegant approach of [Brafman and Tennenholtz, 2003], which finds a near-optimal policy while requiring only a reasonable number of trajectories from the MDP. We call the algorithm R-MAX-E, which is simply an episodic modification of the R-MAX algorithm in Brafman and Tennenholtz [2003]. The algorithm, described in Algorithm 1 utilizes the same principle of *optimism in the face of uncertainty* which inspires the UCB methods we have seen before. Concretely, the algorithm maintains an estimate of the transition probabilities $\Gamma(s'|s,a)$ for all the neighbors s' of a state s, given an action a. It also estimates the reward $\mathbb{R}(s,a)$. Once the algorithm has visited s adequately often to ensure that these estimates are all accurate, it declares the state as *known*. Learning is complete when all the states are known. During a run of the algorithm, whenever it is in a known state, it already knows the optimal action to take and follows this action. However, when the algorithm is in an unknown state, it explores by picking the action chosen least often in the state so far.

Algorithm 1 R-MAX-E algorithm for sample efficient epsisodic reinforcement learning

```
Require: Parameter $m$ to set known state.
  Initialize the set of known states $K = \emptyset$, counters $n_1(s) = n(s,a) = n(s,a,s') = 0$ for all $s,a,s' \in \mathcal{S}\times\mathcal{A}\times\mathcal{S}$
  and $R(s,a) = 0$.
  for all episodes $i = 1,2,3,\ldots$ do
      Let $\widehat{M}$ have $\widehat{\nu}(s) = n_1(s)/i$, $\widehat{\Gamma}(s'|s,a) = n(s,a,s')/n(s,a)$ and $\widehat{\mathcal{R}}(s,a) = R(s,a)/n(s,a)$.
      Let $\widehat{M}_K$ be the induced MDP (see Definition 10.3) and $\pi_i = \pi^\star(\widehat{M}_K)$ be the optimal policy in $\widehat{M}_K$.
      for $t = 1,2,\ldots,H$ do
          Observe state $s_t$
          If $t = 1$, then $n_1(s) += 1$
          If $s_t \in K$, choose $a_t = \pi^\star(s_t, \widehat{M}_K)$, else $a_t = \arg\min_{a\in\mathcal{A}} n(s,a)$
          Receive reward $r_t$
          If $s_t \notin K$, update $n(s_t,a_t) += 1$ and $R(s_t,a_t) += r_t$
          If $s_{t-1} \notin K$, update $n(s_{t-1},a_{t-1},s_t) += 1$
      end for
      If a state becomes known, i.e. $n(s,a) \geq m$ for all $a \in \mathcal{A}$, update $K = K \cup \{s\}$.
  end for
```

In order to more concretely discuss the algorithm, we need some important definitions. Given an MDP M and a set K of known states, we next define the notion of an *Induced MDP*.

Definition 10.3 (Induced MDP) *Let M be an MDP parametrized by $(\mathcal{S},\mathcal{A},\mathcal{R}_M,\Gamma_M,\nu,H)$ with $K \in \mathcal{S}$ being a subset of states. Based on this set, we define the* induced MDP *M_K parametrized by $(\mathcal{S},\mathcal{A},\mathcal{R}_{M_K},\Gamma_{M_K},\nu,H)$ in the following manner. For each $s \in K$, we define*

$$\Gamma_{M_K}(s'|s,a) = \Gamma_M(s'|s,a) \quad \textit{and} \quad \mathcal{R}_{M_K}(s,a) = \mathcal{R}_M(s,a).$$

For all the $s \notin K$, we define

$$\Gamma_{M_K}(s'|s,a) = \mathbf{1}(s' = s) \quad \textit{and} \quad \mathcal{R}_{M_K}(r|s,a) = \mathbf{1}(r = 1).$$

Thus, an induced MDP given a set K of known states is an optimistic process where we receive a reward of 1 (recall that the rewards $r_t \in [0,1]$ so that 1 is the largest attainable reward) no matter which action we try. Furthermore, once we enter such an unknown state, we stay there and keep collecting this reward for the remainder of the episode. On the known states, naturally the transition and reward distributions follow their known behavior.

Given the algorithm gives us some clues as to how exploration in RL is more sophisticated than the exploration in CB. While we do explicitly perform less frequently chosen actions on unknown states as in bandits, we also choose actions according to the policy π_i in the known states. When we are in a known state, π_i is still likely to favor actions which lead to unknown states as those states have high values owing to our optimism. Consequently, exploration in RL is not only about trying all actions given a state the agent is in, but systematically executing policies which lead to the unknown parts of the state space, possibly over a long sequence of actions.

Note that R-MAX-E requires computing the optimal policy $\pi^\star(\widehat{M}_K)$, which can be done using the value iteration scheme from the previous section since the transitions and rewards of $\widehat{M}_K$ are fully known.

We will now analyze the R-MAX-E algorithm, and provide a bound on the number of episodes before which it finds an ϵ-optimal policy. We will prove the following theorem.

Theorem 10.4 *For any $0 \le \epsilon, \delta < 1$, with probability at least $1 - \delta$, the policy π_i computed in episode i of the* R-MAX-E *algorithm satisfies $V(\pi_i, M) \ge V(\pi^\star, M) - \epsilon$ once i is at least $\mathcal{O}\left(\frac{H^4 S^3 A}{\epsilon^3} \log^2 \frac{S^2 A}{\delta}\right)$.*

In order to prove the result, we will inttroduce a number of key concepts in understanding exploration in reinforcement learning. We will present details of the new ideas, while discussing the more technical parts at a high-level. Throughout the analysis, we will abuse our notation $\mathcal{R}$ to also refer to the expected reward, given a (state, action) pair.

We start with a basic result which was first introduced in Kearns and Singh [2002] under the name of a simulation lemma.

Lemma 10.5 (Simulation lemma for episodic MDPs) *Let M and M' be to MDPs with the same state and action spaces. If the transition and reward functions of these MDPs satisfy*

$$\sum_{s' \in \mathcal{S}} |\Gamma_M(s'|s,a) - \Gamma_{M'}(s'|s,a)| \le \epsilon_1, \quad \forall s \in \mathcal{S} \textit{ and } a \in \mathcal{A}, |\mathcal{R}_M(s,a) - \mathcal{R}_{M'}(s,a)| \le \epsilon_2 \quad \forall s \in \mathcal{S} \textit{ and } a \in \mathcal{A}.$$

Then for every non-stationary policy π, the two MDPs satisfy $|V(\pi, M) - V(\pi, M')| \le H\epsilon_1 + \frac{H(H-1)}{2}\epsilon_2$.

The lemma is called a simulation lemma as it tells how much error we incur in evaluating policies if we build an approximate simulator M' for the true process M.

Proof: We will show this lemma using the inductive hypothesis that for any state s, policy π and step $t = 1, 2, \ldots, H$, we have

$$|V_t^\pi(s, M) - V_t^\pi(s, M')| \le (H - t)\epsilon_1 + \frac{(H-t)(H-t+1)}{2}\epsilon_2.$$

At the step $t = H$, this is clearly true since the two value functions only differ in the reward at the final step, which can vary by at most ϵ_2. This establishes the base case.

For the inductive step, we note that by the recursive formuation (10.4), we have

$$
\begin{aligned}
|V_t^\pi(s,M) - V_t^\pi(s,M')| &= \left|\mathbb{E}_{r,s'\sim M,\pi}[r + V_{t+1}^\pi(s',M)] - \mathbb{E}_{r,s'\sim M',\pi}[r + V_{t+1}^\pi(s',M')]\right| \\
&\le \epsilon_2 + \left|\sum_{s'\in\mathcal{S}} \left(\Gamma_M(s'|s,a)V_{t+1}^\pi(s',M) - \Gamma_{M'}(s'|s,a)V_{t+1}^\pi(s',M')\right)\right| \\
&= \epsilon_2 + \left|\sum_{s'\in\mathcal{S}} \left(\Gamma_M(s'|s,a)V_{t+1}^\pi(s',M) - \Gamma_{M'}(s'|s,a)V_{t+1}^\pi(s',M)\right)\right| \\
&\quad + \left|\sum_{s'\in\mathcal{S}} \left(\Gamma_{M'}(s'|s,a)V_{t+1}^\pi(s',M) - \Gamma_{M'}(s'|s,a)V_{t+1}^\pi(s',M')\right)\right| \\
&\le \epsilon_2 + (H-t)\epsilon_1 + (H-t-1)\epsilon_2 + \frac{(H-t-1)(H-t)}{2}\epsilon_1 \\
&= (H-t)\epsilon_2 + \frac{(H-t)(H-t+1)}{2}\epsilon_1.
\end{aligned}
$$

Here the first inequality follows from triangle inequality and using the assumption on the reward functions between M and M'. The second inequality follows by noting that $V_t^\pi(s',M) \le H-t$, since the individual rewards are in $[0,1]$ so that the maximum cumulative reward from time t onwards is at most $H-t$. ■

The next lemma really formalizes our intuition that the optimal policy in the induced MDP encourages exploration of the currently unknown states. We will show that either the best policy π_i learned using the induced MDP at an epsisode i is already good, or it has a high chance of taking us to an unknown state. We will use the notation $\mathbb{P}_M^\pi[\text{escape from } K \text{ in an episode}|s_1 = s]$ to denote the probability of generating a trajectory $(s, a_1, r_1, s_2, a_2, r_2, \ldots, s_H, a_H, r_H)$ where $a_t \sim \pi_t(s_t)$ and $s_t \notin K$ for some t.

Lemma 10.6 (Induced inequalities) *Let M be an MDP with K being the set of known states. Let M_K be the induced MDP (Definition 10.3) with respect to K and M. For any non-stationary policy π and state $s \in \mathcal{S}$ we have*

$$V^\pi(s,M_K) \ge V^\pi(s,M) \quad \textit{and} \quad V^\pi(s,M) \ge V^\pi(s,M_K) - \mathbb{P}_M^\pi[\textit{escape from } K \textit{ in an episode}|s_1 = s].$$

The lemma has two implications. First it formalizes the notion that the induced MDP M_K is indeed an optimistic version of M, since it ascribes higher values to each state under *every policy*. At the same time, the optimism is not uncontrolled. The values ascribed by M_K to a policy π are higher only if the policy has a substantial probability of visiting an unknown state, and hence is useful for exploration.

Proof: The first inequality is a direct consequence of the definition of M_K. If $s \notin K$, it is immediate since we get the maximum reward of 1 at each time-step, while never leaving this state. If $s \in K$, then our immediate reward is identical to that in M. At the next step, we either stay in K, or leave. If we leave then we will obtain the largest reward for the remaining time steps. If we stay, we obtain the same reward as that in M. Thus we never obtain a smaller reward in M_K by definition.

We only sketch the proof of the second inequality as the intuition is clear and the rest is just algebra. According to our argument above, if π never leaves the known states, then the value functions in M and M_K should coincide. Consequently, the difference in value functions can be bounded by the probability of leaving the set of known states at some point in the trajectory, which is exactly what the lemma says. ■

Given the lemma, we have a particularly useful corollary. It says that the policy π_i computed in each episode of Algorithm 1 is near optimal, with the error being the probability of leaving the known state set.

Corollary 10.7 (Implicit Explore-Exploit)

$$V^{\pi^\star(M_K)}(s, M) \geq V^\star(s, M) - \mathbb{P}_M^{\pi^\star(M_K)}[\textit{escape from } K \textit{ in an episode}|s_1 = s]$$

Proof: By the lemma, we have

$$\begin{aligned} V^{\pi^\star(M_K)}(s, M) &\geq V^{\pi^\star(M_K)}(s, M_K) - \mathbb{P}_M^{\pi^\star(M_K)}[\text{escape from } K \text{ in an episode}|s_1 = s] \\ &\geq V^{\pi^\star(M)}(s, M_K) - \mathbb{P}_M^{\pi^\star(M_K)}[\text{escape from } K \text{ in an episode}|s_1 = s] \\ &\geq V^{\pi^\star(M)}(s, M) - \mathbb{P}_M^{\pi^\star(M_K)}[\text{escape from } K \text{ in an episode}|s_1 = s]. \end{aligned}$$

Here the first inequality follows from Lemma 10.6 applied with $\pi = \pi^\star(M_K)$, second inequality uses that $\pi^\star(M_K)$ is the optimal policy in M_K and hence obtains a higher reward than $\pi^\star(M)$ and the third inequality follows from the optimism of M_K shown in Lemma 10.6. ■

We now have most of the ingredients for the theorem. In order to prove the theorem, we need to ensure that m is large enough that when a state is declared known, then its transition and reward functions are reasonably accurate. For now, let us assume that m is chosen large enough so that the induced approximate MDP $\widehat{M}_K$ is a good approximation to the true induced MDP M_K. Based on Lemma 10.5, we will assume that m is large enough so that the value functions of these two MDPs are at most $\epsilon/2$ different in any state. Then, applying this closeness twice, we see that

$$V^{\widehat{\pi}_i}(s, M_K) \geq V^{\widehat{\pi}_i}(s, \widehat{M}_K) - \frac{\epsilon}{2} \geq V^{\pi^\star}(s, \widehat{M}_K) - \frac{\epsilon}{2} \geq V^{\pi^\star}(s, M_K) - \epsilon.$$

Combining with Lemma 10.6, we see that for any starting state we have

$$\begin{aligned} V^{\widehat{\pi}_i}(s, M) &\geq V^{\pi^\star}(s, M_K) - \epsilon - \mathbb{P}_M^{\widehat{\pi}_i}[\text{escape from } K \text{ in an episode}|s_1 = s] \\ &\geq V^{\pi^\star}(s, M) - \epsilon - \mathbb{P}_M^{\widehat{\pi}_i}[\text{escape from } K \text{ in an episode}|s_1 = s], \end{aligned}$$

where the first inequality is combining Lemma 10.6 with with the earlier bound, and the second inequality uses the optimism of the induced MDP M_K.

Thus, either the policy $\widehat{\pi}_i$ is at most 2ϵ suboptimal, or it visits an unknown state with probability at least ϵ. Intuitively, this means that we visit an unknown state at least every $1/\epsilon$ steps, if $\widehat{\pi}_i$ is not already near optimal. The total number of visits to unknown states are bounded by mSA. This is because for each unknown state s, we need to try every action a at least m times before s becomes known. Since we try the least frequently chosen action each time, it is ensured that each action is chosen *exactly m times* before s becomes known. Consequently, we need roughly mSA/ϵ episodes in order to ensure that every state is known and the algorithm can certifiably have a near optimal policy.

In order to obtain the theorem statement, we set $m = \mathcal{O}\left(\frac{S^2H^4}{\epsilon^2}\log\frac{S^2A}{\delta}\right)$. Based on some calculations, which can be found in either Brafman and Tennenholtz [2003] or Kearns and Singh [2002], it can be shown that this value of m suffices for $\widehat{M}_K$ to be a good approximation for M_K in that their value functions are close by Lemma 10.5.

References

Ronen I. Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2003.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

Martin Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.