Stochastic Dynamic Optimisation

Connie Trojan 4 May, 2022

1 Introduction

This report will discuss solution methods and properties of Markov decision processes (MDPs) and related decision-making processes. This first section will focus on defining the Markov decision process and its objectives, as well as introducing our two main running examples. Section 2 will introduce some solution methods for MDPs whose dynamics are known, while Section 3 will focus on methods that do not require this assumption and instead learn solely by interacting with the environment. We will consider an extension to the problem where there are multiple decision-making agents in Section 4. Finally, Section 5 will discuss some active and open research areas in the field.

1.1 Markov Decision Processes

A **Markov decision process** is a sequential decision-making process. At each time step, the decision maker or **agent** finds themself in some state S from a finite **state space** $\mathcal{S}$ and must select some action A from a finite action set $\mathcal{A}(S)$. After taking an action, the process moves to the next time step, transitioning randomly to some new state S' according to fixed transition probabilities $\mathbb{P}(S' \mid S, A)$ and awarding a reward $R(S, A)$ to the agent. The reward might also be stochastic, in which case $R(S, A)$ can be used to refer to the expected immediate reward from choosing A in state S.

The main objective in studying MDPs is to find the decision rule or **policy** $\pi : \mathcal{S} \to \mathcal{A}$ that maximises expected reward in some sense. In a task with a fixed termination time, or where the process will eventually enter a terminal state and end (for example, a game where rewards are given on termination according to whether the player won or lost), it is natural to maximise the total reward obtained since this is always finite. Such tasks are known as **episodic**. For tasks that can continue indefinitely or that will end at an unknown or very distant time point, it will be convenient (or necessary) to consider the time horizon to be infinite. In this case the total reward earned can often also be infinite, so that "maximising total reward" is no longer a well defined objective. In this case a natural alternative objective might be to maximise the rate at which reward is accumulated via the long run **limiting average reward**:

$$\lim_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} R(S_t, A_t)\,. \tag{1}$$

Another possible objective is to maximise the total reward when a discount factor of $\gamma \in [0, 1)$ is applied to future rewards for each time period until they are earned:

$$\sum_{t=0}^{\infty} \gamma^t \, R(S_t, A_t) \, . \tag{2}$$

This is known as the **discounted reward**. The use of discounting might be motivated by a mechanism like inflation, by which rewards earned in the future are less valuable than those earned immediately. It is also mathematically equivalent to having some constant probability $1-\gamma$ at each time step of the process being interrupted or entering some zero-reward absorbing state (Watkins (1989)). The discounted reward formulation is more often used than the limiting average for mathematical convenience, since it is easier to prove results in this setting. In addition, both the total and limiting average reward formulations can be linked to the discounted case. For episodic MDPs, we can consider terminal states to be absorbing with 0 reward and allow γ to be 1. The limiting average reward can be considered to be obtained in the limit as $\gamma \to 1$ - in fact, a solution to the limiting average case can always be obtained by solving the discounted case for γ sufficiently close to 1. More precisely, a theorem proved by Filar and Vrieze (1997) states that for every MDP there exists $\gamma_0 \in [0, 1)$ such that the optimal deterministic policy π_0 for the γ_0-discounted MDP is also optimal for the γ-discounted MDP for all $\gamma \in [\gamma_0, 1)$ and the limiting average formulation.

1.2 Blackjack

We will use the following simple formulation of the card game blackjack as an episodic Markov decision process as a running example (Sutton and Barto (2018)). The game begins with the dealer dealing a card to each player and themself. The cards have the following values: 2-10 are worth their face values, picture cards (jack, queen, king) are worth 10, and aces can be counted as either 1 or 11. On their turn, players have two choices: **hit** (be dealt another card from the deck) or **stick** (stop drawing cards). If the total value of a player's cards exceeds 21 they **go bust** and immediately lose. After all players have opted to stick, the dealer plays according to a fixed strategy where they hit on any total up to 16 and stick on 17 or above. All remaining players win if the dealer goes bust. Otherwise, remaining hands win if their total value is higher than the dealers, draw if it is the same, or lose if it is lower.

Assuming that the cards are dealt from a deck that is sufficiently large that the probabilities of drawing any particular card do not change significantly over the course of the game, draws from the deck can be modelled as i.i.d. and there is no benefit to keeping track of the history of cards dealt. This means that each player can be considered to play independently against the dealer, and the state space is defined by the player's current total (with an ace counted as 11 unless doing so results in a total over 21), the dealer's card, and whether or not the player is currently counting an ace as an 11. We also have three possible terminal states: `WIN`, `DRAW`, and `LOSE`, with the player receiving a reward of 1, 0, or -1 respectively on transition.

We will find the optimal blackjack strategy exactly using dynamic programming in Section 2.5. In Section 3.3 we will use this for a comparison of methods that only require interaction with the environment, and hence can be used to learn to play from experience only without exact knowledge of the system dynamics (for example, they could be used if the composition of the deck is unknown).

1.3 Patrolling Problems

In patrolling problems, a **patroller** must choose a patrol route between locations in order to detect **attacks**. This framework applies to many settings and definitions of 'attack' (for example, a safety inspector might consider breaches of regulations to be attacks), including those where the 'patrol route' is not be a physical path (for example, for a security officer deciding which CCTV feed to monitor next).

This was formulated as a Markov decision process by Lin et al. (2013) - in their formulation, the area to be patrolled is a graph where each node is considered to be a potential attack location and edges exist between nodes that the patroller can travel between in a single time period. Given that attackers arrive randomly at each node at known constant rates and require a random amount of time to complete their attacks, the patroller must sequentially decide which node to visit next. When arrivals are modelled by homogenous Poisson processes, nothing can be learned from past attacks about future attack times, so the patroller's state can be identified as the number of time periods elapsed since the patroller last visited each node. The patroller's objective is to minimise the limiting average cost incurred by undetected attacks.

In the case where the graph is not fully connected, the patroller must balance visiting adjacent nodes where costly attacks are more likely to be taking place with visiting nodes that enable travel to other areas of the graph. Lin et al. (2013) solve this problem exactly using linear programming (which we will consider in Section 2.4) and suggest an approximate index heuristic solution for cases where this is computationally intractable (see Section 2.6).

Lin et al. (2014) discuss an extension to this problem where detection is imperfect and the patroller has a fixed probability at each node of detecting any ongoing attacks. This can also be seen as an MDP, but with a more complex state space - assuming attack times are bounded above by some fixed upper bound B, the patroller must recall the history of nodes visited in the last B time periods in order to compute the expected cost of not visiting each node.

In practical problems, it might make sense to consider attackers to be strategic, seeking to enact a strategy that avoids detection and maximises damage done. In Section 4.5 we will look at some formulations of the patrol problem that can be seen as competitive MDPs.

2 Dynamic Programming

This section will focus on methods for finding the optimal solution to MDPs when the full system dynamics are known. These are typically derived from the **Bellman optimality equations** and require calculations over the whole state-action space. The ideas introduced in this section will also inform many of the methods that will be introduced in Section 3 for cases when the dynamics of the environment are unknown or exact solution methods are computationally intractable.

2.1 The Bellman Equations

We can define the value $v_t^\pi(S)$ of being in a particular state at time t under policy π as the expected reward obtained from following policy π starting from state S at time t:

$$v_t^\pi(S) = R(S, \pi(S)) + \gamma \mathbb{E}\left(v_{t+1}^\pi(S') \,|\, S, A_t = \pi(S)\right) . \tag{3}$$

Under an optimal policy (where $v_t^\pi(S)$ is maximal for each state), the action with maximal expected reward is chosen at each time step, so that the values under an optimal policy satisfy:

$$v_t^{\pi_{opt}}(S) = \max_{A_t \in \mathcal{A}(S)} \left\{R(S, A_t) + \gamma \mathbb{E}\left(v_{t+1}^{\pi_{opt}}(S') \,|\, S, A_t\right)\right\} . \tag{4}$$

That is to say, the value of being in state S is the expected value of the immediate reward received plus the value of the state we transition to, given that we take the action A realising the maximum in Equation 4. These state values will be the same for any optimal policy, so the value of being in state S at time t and playing optimally can be uniquely defined as $v_t(S)$.

If there is a fixed time horizon T then these state values can be time-dependent. If the terminal rewards $v_T(S)$ are known for each possible terminal state, then the system in Equation 4 can be solved exactly by backwards recursion (this process called **backwards dynamic programming**), and an optimal policy is one that chooses an action realising the maximum at each time.

In the infinite horizon setting, it is always possible to find a **pure stationary** optimal policy, i.e. one that always selects the same action in each state (Powell (2011)). This means that the value of each state does not depend on time, so that we can drop the time indexing from Equation 4 to obtain the **Bellman optimality equations**:

$$\begin{aligned} v(S) &= \max_{A \in \mathcal{A}(S)} \left\{R(S, A) + \gamma \mathbb{E}\left(v(S') \,|\, S, A\right)\right\} \\ &= \max_{A \in \mathcal{A}(S)} \left\{R(S, A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}\left(S' \,|\, S, A\right) v(S')\right\} . \end{aligned} \tag{5}$$

The $v(S)$ are uniquely defined by this system of equations, and an optimal stationary policy takes value $\pi_{opt}(S) \in \text{argmax}_{A \in \mathcal{A}(S)} R(S, A) + \gamma \mathbb{E}\left(v(S') \,|\, S, A\right)$.

2.2 Value Iteration

One of the simplest algorithms for solving MDPs is known as **value iteration** (Algorithm 1), a process that produces a sequence of estimates $v^{(n)}$ of the $v(s)$ by iteratively applying the Bellman equations as an update rule. Performing n iterations of value iteration is equivalent to using backwards dynamic programming to find the state values at time $t = 0$ in a finite horizon problem with $T = n$ time periods that gives reward $v_T(S) = v^{(0)}(S)$ for terminating in state S. This means that we are finding the $v(S)$ as a limit of the $v_0(S)$ in a finite horizon problem as the time horizon is increased. This sequence of approximations converges uniformly to the true state values as $T \to \infty$ (Ross (1995)).

Algorithm 1: Value Iteration

Result: estimated value function v^ϵ, estimated optimal policy π^ϵ

Require: tolerance $\epsilon > 0$

Initialise: $v^{(0)}(S) = 0 \quad \forall S \in \mathcal{S}$

$n \leftarrow 0$;

do

$n \leftarrow n + 1$;

$v^{(n)}(S) \leftarrow \max_{A \in \mathcal{A}(S)} \left\{ R(S, A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S' \mid S, A) v^{(n-1)}(S') \right\} \quad \forall S \in \mathcal{S}$;

while $\|v^{(n)} - v^{(n-1)}\|_\infty \geq \epsilon(1 - \gamma)/2\gamma$;

$v^\epsilon \leftarrow v^{(n)}$;

$\pi^\epsilon(S) \leftarrow \text{argmax}_{A \in \mathcal{A}(S)} \left\{ R(S, A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S' \mid S, A) v^\epsilon(S') \right\} \quad \forall S \in \mathcal{S}$;

Convergence can be sped up by replacing $v^{(n-1)}(S')$ with $v^{(n)}(S')$ for the states whose values have already been updated when looping over states to update the value function - this is called the **Gauss-Seidel** variant of the algorithm (Powell (2011)). In cases where it is the optimal policy and not the value function that is of interest, it may be faster to focus on convergence of the difference between state values, since this is all that is needed to find the optimal policy (Powell (2011)). This variant is known as **relative value iteration**, and involves adding the additional update $v^{(n)}(S) \leftarrow v^{(n)}(S) - v^{(n)}(\tilde{S})$ to each iteration, for some arbitrary fixed reference state $\tilde{S}$. This variant can be used to solve the limiting average version of the MDP, by setting $\gamma = 1$ and changing the stopping condition to $\|v^{(n)} - v^{(n-1)}\|_\infty \geq \epsilon$. In this case an analogous uniform convergence result was proved by White (1963) for the new sequence of $v^{(n)}$.

2.3 Policy Iteration

Suppose we already have some policy π, for which we have computed v^π. How could we improve on π? If we had the option of taking an action other than $\pi(S)$ and then using policy π at each subsequent timestep, then the best choice would be an action $\pi'(S) \in \text{argmax}_{A \in \mathcal{A}(S)} R(S, A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S' \mid S, A) v^\pi(S')$ that maximises the expected reward given that

each state's future value is v^{π}. π' defines a new policy, that in fact is at least as good as π and as long as π was not already optimal, is strictly better for some initial states. This can be proved by induction (Ross (1995)): since using π' for one decision and using π thereafter is better than simply using π, it follows that using π' for n timesteps (and π thereafter) is also better than following π, for any $n \in \mathbb{N}$. This update is known as **policy improvement**.

If we then evaluate $v^{\pi'}$, the procedure described above can be used iteratively until an optimal policy is found. Convergence is guaranteed in a finite number of iterations since there are a finite number of stationary strategies when the state space is finite (Ross (1995)). This is known as the **policy iteration** algorithm, detailed in Algorithm 2. Note that the exact value of the current policy is calculated at each iteration by solving the vectorised version of Equation 3, where the one-step transition matrix is given by $[P^{(\pi,n)}]_{S,S'} = \mathbb{P}\left(S' \,|\, S, A = \pi(S)\right)$ and the reward vector by $r_S^{(\pi,n)} = R(S, \pi(S))$.

Algorithm 2: Policy Iteration

Result: value vector v, optimal policy π

Require: initial policy $\pi^{(0)}$.

$n \leftarrow 0$;

do

  Compute the one-step transition matrix for policy $\pi^{(n)}$, $P^{(\pi,n)}$;

  Compute the reward vector for policy $\pi^{(n)}$, $r^{(\pi,n)}$;

  $v^{(n)} \leftarrow (I - \gamma P^{(\pi,n)})^{-1}\, r^{(\pi,n)}$;

  $\pi^{(n+1)}(S) \leftarrow \text{argmax}_{A \in \mathcal{A}(S)} \left\{ R(S,A) + \gamma [P^{(\pi,n)}\, v^{(n)}]_S \right\} \quad \forall S \in \mathcal{S}$;

  $n \leftarrow n + 1$;

while $\pi^{(n)} \neq \pi^{(n-1)}$;

$v \leftarrow v^{(n-1)}$;

$\pi \leftarrow \pi^{(n-1)}$;

Note that the $v^{(n)}$ can be seen as a sequence of estimates for the true state value vector as the value of the current estimate $\pi^{(n)}$ of the optimal policy. Each iteration of policy evaluation is actually equivalent to applying an update of Newton's method to the problem of minimizing the L^2 norm of the difference between the left and right hand sides of the vectorised form of the Bellman optimality equations (Filar and Vrieze (1997)).

Policy iteration tends to converge in fewer iterations than value iteration, however the policy evaluation step requires a matrix inversion which is costly for large state spaces, so that each iteration is more computationally intensive (Powell (2011)). The cost of each iteration could be reduced by replacing the policy evaluation step with an approximation of the value of the current policy instead - this idea of alternating approximation of policy evaluation with a step of policy improvement will appear again later in Section 3.

2.4 Linear Programming

It is also possible to formulate the $v(S)$ as the solution to a linear program (Ross (1995)). Since the $v(S)$ satisfy the Bellman optimality equations, they must also satisfy the following set of inequalities for every $S \in \mathcal{S}$:

$$v(S) \geq R(S,A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}(S' \mid S, A) v(S') \qquad \forall A \in \mathcal{A}(S)\,, \tag{6}$$

with equality for the $A = \pi^*(S)$ achieving the maximum in the Bellman equations. v is also the smallest value vector to satisfy the above for all states, in the sense that if u also satisfies 6 then $u(S) \geq v(S)\ \forall S \in \mathcal{S}$. This means we can find all of the state values exactly by solving the following linear program:

$$\min_{u \in \mathbb{R}^{|\mathcal{S}|}} \sum_{S \in \mathcal{S}} u(S), \tag{7}$$

subject to:

$$u(S) \geq R(S,A) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}\left(S' \mid S, A\right) u(S') \qquad \forall A \in \mathcal{A}(S),\ S \in \mathcal{S} \tag{8}$$

The LP can be solved with any standard method, like the simplex algorithm. This is an efficient solution method for small state-action spaces, but quickly becomes computationally impractical for larger problems, as even moderately sized MDPs can result in a number of constraints considered very large for a linear program.

2.5 Blackjack Example

We will now return to the blackjack example described in Section 1.2. Recall that the state space consists of tuples (s, d, a) representing the player total s, dealer card d and whether the player has a usable ace ($a = 1$) or not ($a = 0$), with terminal states `WIN`, `DRAW`, and `LOSE`. At each time step, the player can choose to either hit or stick. If the player decides to hit, drawing a card of value c, the possible state transitions are as follows: If $s \leq 10$, transition to $(s+c, d, a)$ (with $c = 11$ and $a = 1$ for an ace). If $s \geq 11$ and $c \leq 21 - s$, transition to state $(s + c, d, a)$ (with $c = 1$ for an ace). If $c > 21 - s$ and $a = 1$, use the ace and transition to $(s + c - 10, d, 0)$. If $c > 21 - s$ and $a = 0$, go bust and transition to `LOSE`.

If the player sticks on total s, they transition to `WIN` if the dealer goes bust or sticks on total less than s, `DRAW` if they stick on total of s, or `LOSE` if they stick on total greater than s. We also need the transition probabilities for the stick action before dynamic programming can be used. These can be computed from the probability distribution $\mathbb{P}\left(d_T \mid d_I\right)$ over the dealer's possible terminal states $d_T \in \{17, 18, 19, 20, 21, \texttt{BUST}\}$ and starting cards d_I. Since the dealer's strategy is known (hit on totals less than 17), these can be calculated recursively by considering the probabilities starting from each possible intermediate state for the dealer and partitioning on the outcomes of hit actions.

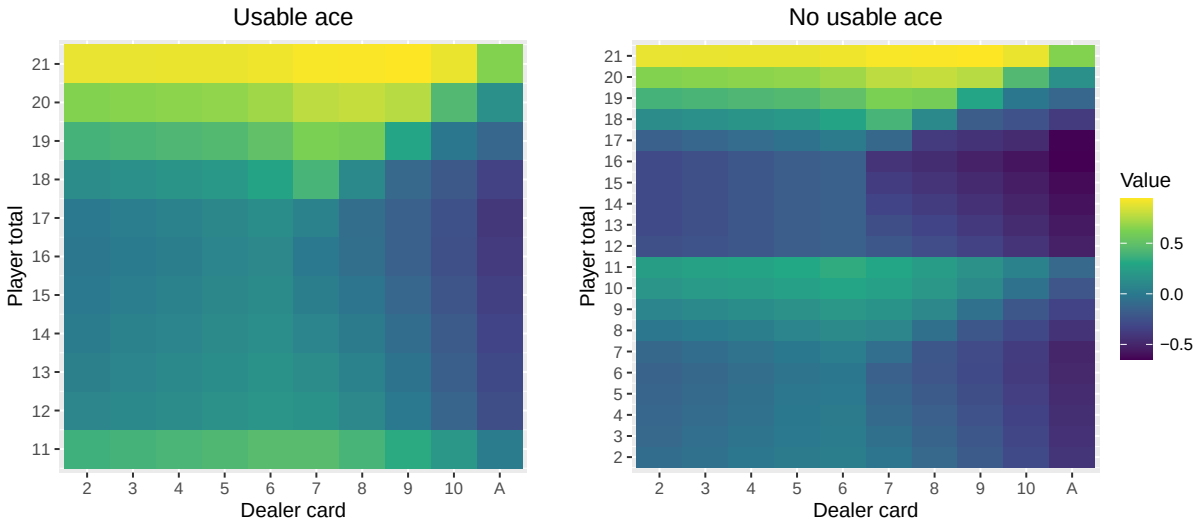

Figure 1: State values in blackjack.

Finally, the exact state values can be found efficiently by solving the linear program formulated in Section 2.4 since the state and action spaces are relatively small, $|\mathcal{S}| = 344$ and $|\mathcal{A}| = 2$. These are shown in Figure 1, and the resulting optimal strategy is represented in Figure 2 (note that it is always optimal to hit on a score less than or equal to 11). Since the distribution over possible starting states is known, it is also possible to calculate the value of the game itself as the expected value of the starting state. In this case it is -0.047, so one can expect to lose money even playing optimally. This is largely due to the fact that if the player goes bust they lose immediately regardless of whether the dealer would have also gone bust, had they taken their turn.

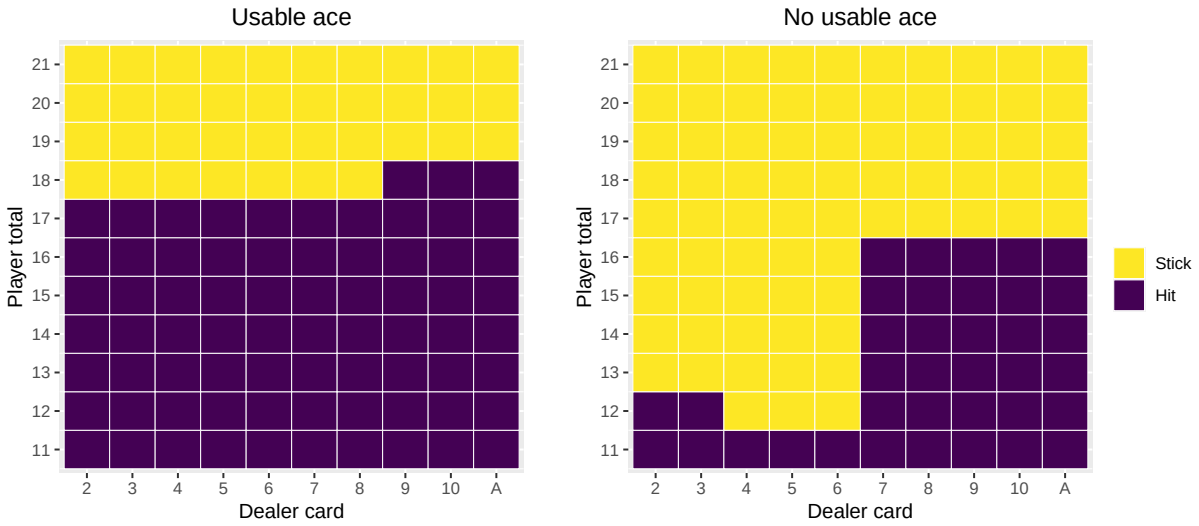

Figure 2: Optimal blackjack strategy.

2.6 Heuristic Policies for the Patrolling Example

The methods introduced so far can be very computationally costly for problems with large state spaces. In such cases, it may be preferable to find a heuristic solution if it is possible to efficiently compute one that is close to optimal. For example, the police patrolling problem described in Section 1.3 was solved approximately by Lin et al. (2013) using an index heuristic policy, since it is very costly to solve exactly for large graphs. Their approach works by computing a **Whittle index** for each node representing the value of visiting it next given the current system state. For complete graphs (where every node is accessible from any other in one timestep), the patroller can simply choose to visit the node with the highest index next.

The Whittle indices are calculated by first formulating a relaxation of the problem, allowing policies to choose multiple nodes to visit next in each state but constraining the total rate at which nodes are visited to be less than or equal to 1. Next, the Lagrange relaxation to this new problem is formulated, moving the average rate constraint to the objective function with a Lagrange multiplier. This relaxation can be broken up into a separate linear program for each node, where the Lagrange multiplier can be interpreted as a charge for visiting that node. The Whittle index $W(k)$ for each node is a function of the number of time periods k since it was last visited, representing the per-visit cost that makes that node indifferent between being visited every k time periods and every $k+1$ time periods.

When the graph is not complete, we also have to consider impact of the choice on future options. Lin et al. (2013) suggest computing a new index heuristic that looks at possible l-step paths starting from each node and aggregates the Whittle indices for the nodes visited, so as to reward visiting nodes that provide a route to others with high Whittle indices. They also suggest computing a similar heuristic which instead aggregates the Whittle indices of unvisited nodes as a penalty. They found that the latter approach performed better, since it is sometimes possible to game the reward heuristic by delaying a visit to a particular node so as to collect a higher reward later on, even if the nodes visited in the meantime do not benefit much from a patrol visit.

3 Reinforcement Learning

This section will focus on **reinforcement learning** (RL), also known as **approximate dynamic programming**. In this framework we no longer assume complete knowledge of the system dynamics, learning only from experience gained from interacting with the system. Usually, this is done by learning an action-value function $Q(S, A)$ representing the expected reward if action A is taken in state S. This implicitly defines the optimal policy as $\pi^*(S) = \text{argmax}_{A \in \mathcal{A}(S)} Q(S, A)$. This section will focus on **tabular** methods, which store estimated Q-values in a lookup table with entries for each possible state-action pair.

3.1 Monte Carlo Control

The idea behind Monte Carlo control is to estimate Q-values for episodic tasks statistically, by averaging observed sample rewards. An approximate version of the policy iteration algorithm from Section 2.3 would be to replace the exact policy evaluation step with an approximate evaluation using Monte Carlo sampling - simulating a number of episodes starting in each state and following the current policy $\pi^{(n)}$, and estimating the state values under $\pi^{(n)}$ by the average of the observed returns. In the RL setting, we could instead start each episode in a particular state-action pair in order to estimate their values if $\pi^{(n)}$ is followed afterwards, and use these to perform a step of policy improvement.

To get a close approximation to each step of policy iteration in this way, it would be necessary to perform a large number of simulations for each state-action pair at each iteration - a computationally costly exercise that couldn't realistically be used if observations are obtained from real-world interaction rather than simulation. An alternative approach is to instead view the values (or state-action values) in policy iteration as a sequence of approximations to the value function under the optimal policy - we could produce such a sequence with Monte Carlo sampling by updating the estimates after each simulated episode. If each state-action pair has a nonzero probability of being chosen to start each episode (this is called the assumption of **exploring starts**), each state-action pair is guaranteed to be visited infinitely often regardless of the policy used, and we will have convergence to the optimal policy (Sutton and Barto (2018)). Algorithm 3 provides the details for such a Monte Carlo control algorithm, where the returns after the first visits to each state-action pair are used to update their Q-values.

Algorithm 3: Monte Carlo Control (Exploring Starts)

Result: table of estimated Q-values Q, estimated optimal policy π

Require: number of episodes E, initial policy π

Initialise: $R(S,A),\ n(S,A) = 0 \quad \forall A \in \mathcal{A}(S),\ S \in \mathcal{S}$

for $e = 1, \ldots, E$ **do**

  Initialise $S_0 \in \mathcal{S},\ A_0 \in \mathcal{A}(S)$;

  Observe $R_0, (S_1, A_1, R_1), \ldots, (S_T, A_T, R_T) \leftarrow$ `MDP_episode`(S_0, A_0, π) ;

  $G \leftarrow 0$

  for $t = T, \ldots, 0$ **do**

    $G \leftarrow G + \gamma R_t$;

    if $(S_t, A_t) \notin \{(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})\}$ **then**

      $R(S_t, A_t) \leftarrow R(S_t, A_t) + G$;

      $n(S_t, A_t) \leftarrow n(S_t, A_t) + 1$;

      $Q(S_t, A_t) \leftarrow R(S_t, A_t)/n(S_t, A_t)$;

      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$;

  end

end

3.2 Q-Learning

The MC algorithm described in the previous section was **on-policy**: the policy used to interact with the environment was the one that was optimal according to the current estimated table of Q-values. This was why the assumption of exploring starts was needed to guarantee convergence to the true optimal policy, as the algorithm would otherwise never choose actions considered suboptimal by the current estimate of the Q-value table. If we need to perform inference based on interaction with a real-world system, it is likely that we will not be able to guarantee this since the initial conditions may be outside of our control. An alternative approach known as **off-policy** is to use one policy (the **behaviour policy**) to explore the state-action space and observe rewards, while using the updated Q-values to maintain a separate estimate of the true optimal policy (the **target policy**).

One example of an off-policy RL algorithm is **Q-learning**, first described by Watkins (1989). Any behaviour policy can be used to interact with the environment. Any time an action A is taken in state S and the immediate reward R and new state S' are observed, the following update is performed:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', \pi(S)) - Q(S, A)) \,. \tag{9}$$

Here, $\alpha \in (0, 1]$ is a hyperparameter controlling the size of updates to the Q-values, called the **learning rate**. The estimates converge almost surely to the true Q-values, as long as the behaviour policy always has a non-zero chance of visiting each state-action pair and subject to conditions on the learning rate (Watkins and Dayan (1992)). In particular, α should be reduced in each iteration so that the sequence of rates has infinite sum (so that the true Q-values can be reached regardless of the starting values) but is square-summable (to guarantee convergence). In practise a small, constant learning rate may be easier to use if only an approximation to the optimal policy is needed, since convergence can be very slow if the decay rate of the α is not chosen carefully. The choice of behaviour policy also has an impact on the convergence rate - using policies that balance exploration with exploiting the current best known action causes much faster convergence than purely random selection (Gosavi (2009)). A simple behaviour policy is to use an **epsilon-greedy** policy derived from the current target policy (Sutton and Barto (2018)): fix some small value for $\epsilon \in (0, 1)$ and take with probability $1 - \epsilon$ the action that is optimal under the target policy, else select an action from the action set at random. This version of Q-learning is described in Algorithm 4.

It can be adjusted for the limiting average setting in a similar way to that seen in Section 2.2, by instead considering the Q-values relative to that of a reference state-action pair (Gosavi (2009)). Note that in contrast to the Monte Carlo algorithm described in the previous section, the Q-values and hence the policies are updated during each episode instead of at the end. This means that we aren't actually limited to the episodic case anymore - Q-learning can learn in an online fashion.

Algorithm 4: Q-Learning (ϵ-greedy action selection)

Result: table of estimated Q-values Q, estimated optimal policy π

Require: number of episodes E, learning rate $\alpha \in (0, 1]$, small $\epsilon \in (0, 1)$

Initialise: $Q(S, A) \in \mathbb{R} \quad \forall A \in \mathcal{A}(S),\ S \in \mathcal{S}$, with $Q(S, A) = 0$ for all terminal states

for $e = 1, \ldots, E$ **do**

  $t \leftarrow 0$;

  Initialise $S_0 \in \mathcal{S}$;

  while S_t *is non-terminal* **do**

    With probability $1 - \epsilon$ set $A_t \leftarrow \pi(S_t)$ else select random $A_t \in \mathcal{A}(S)$;

    Observe $S'_t, R_t \leftarrow \texttt{MDP\_step}(S_t, A_t)$;

    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma Q(S'_t, \pi(S'_t)) - Q(S_t, A_t))$;

    $\pi(S) \leftarrow \text{argmax}_a\, Q(S, a) \quad \forall S \in \mathcal{S}$;

    $S_{t+1} \leftarrow S'_t$;

    $t \leftarrow t + 1$;

  end

end

3.3 Blackjack Example

The blackjack example (described in Section 1.2 and solved exactly in Section 2.5) lends itself to use of the RL techniques discussed so far: the state transition probabilities are challenging to compute analytically but easy to simulate, and the state-action space is small enough that storing and learning a table of Q-values is practical. For Monte Carlo control, the exploring starts were implemented by selecting state-actions pairs with equal probability, with an initial policy that always chose the hit action. Q values were all initialised at 0. The values of α and ϵ for Q-learning were set to $\alpha = 0.005$, $\epsilon = 0.04$, chosen by a gridsearch to maximise average reward after half a million training episodes. The performance of Q-learning varied greatly depending on the choice of these hyperparameters, with very slow convergence and erratic behaviour for larger values.

The results for both algorithms over five replications are illustrated in Figure 3, where the mean performance $\pm$ one standard deviation is indicated. Performance was evaluated by computing the average reward over 10000 simulated games. The values of the optimal strategy computed in Section 2.5 and the dealer strategy (stick only on totals of 17 or higher) are also included for reference. Both algorithms quickly find strategies that outperform the dealer strategy, and converge to strategies close to optimal within the million training episodes used. Only MC control was observed to ever find the true optimal strategy within this time.

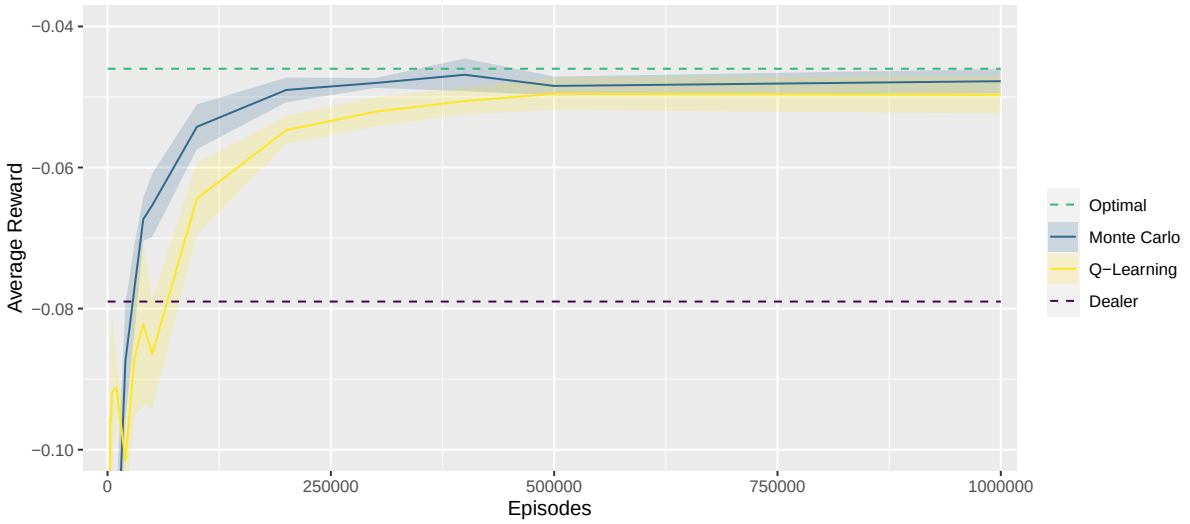

Figure 3: Performance of the tabular RL algorithms on the blackjack example.

4 Stochastic Games

This section focuses on **stochastic games** (SGs), or **competitive MDP**s. Stochastic games are the multi-agent generalisation of MDPs, where there is more than one agent or **player**, and the state transitions and rewards can depend on all of their choices. They can also be seen as the sequential, stochastic generalisation of a **matrix game**. This was the context in which they were first described by Shapley (1953), whose paper predates the first formulation of MDPs.

This section will focus on the existence and nature of optimal policies in this setting, with some solution methods for when the system dynamics are known. It is also possible to apply reinforcement learning to the multi agent setting, although this introduces extra challenges that will be discussed later in Section 5.1. We will start by introducing some ideas from game theory that will help in defining objectives and solution methods for stochastic games.

4.1 Nash Equilibrium

In a 2-player game Γ, each player i can select any action $A^i \in \mathcal{A}^i$. Once all players have made their decision, each receives payoff $R^i(A^1, A^2)$. The key objective in solving such games is to identify **Nash equilibrium** (NE) policies, defined as pairs of strategies π^1, π^2 where neither player can guarantee a better expected outcome by unilaterally switching to a different strategy. Nash equilibria always exist for matrix games, but unless the game is **zero-sum** ($R^1 \equiv -R^2$) players may have different expected payoffs in different Nash equilibria (Peterson (2017)). Note that NE strategies are often **mixed**, defining a distribution over possible actions rather than identifying one action as optimal. For example, consider the two-player zero-sum game with the following payoffs for player 1:

$$\Gamma = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Here, if player 1 always chose action 1 (or 2), player 2 would be able to guarantee a payoff of -1. However, if player 1 randomly chooses either option with probability 1/2 then their expected payoff is 0 regardless of player 2's strategy. If both players do this then we have a NE.

In a two-player stochastic game, each state $S \in \mathcal{S}$ corresponds to a matrix game $\Gamma(S)$ with rewards $R^i(S, A^1, A^2)$ for $A^1 \in \mathcal{A}^1(S), A^2 \in \mathcal{A}^2(S)$. At each timestep, the game corresponding to the current state is played. Rewards are awarded and a state transition to S' occurs according to fixed probabilities $\mathbb{P}\left(S' \mid S, A^1, A^2\right)$ that depend on the choices of the players. Study of stochastic games usually focuses on finding mixed stationary policies that are Nash equilibria - pairs of stationary policies for which neither player can guarantee a better value (expected discounted/limiting average reward) by unilaterally changing their policy. Note that NE policies always exist, but again need not have a unique value unless all of the games have zero-sum rewards (Filar and Vrieze (1997)). In the fully co-operative case, where all players receive the same rewards, single agent MDP solution strategies can be used and the optimal strategy in the MDP will be a Nash equilibrium for the stochastic game (Zhang et al. (2021)).

Nash equilibrium policies for SGs need not be stationary and are usually different to the NE strategy for each individual game, and not only because possible future states must be considered. For example, the famous prisoners' dilemma 2-player game has a unique NE where both players betray the other. However, in the iterated version (where it is repeated indefinitely like a single-state SG) there are many NE strategies with better long-run average reward that involve co-operation - for example, the tit-for-tat strategy where the player co-operates in the first round and mimics their opponent's most recent move thereafter (Peterson (2017)).

4.2 Discounted Rewards

Shapley (1953) proved that Nash equilibrium stationary strategies always exist for two-player zero-sum SGs with positive stopping probabilities in every state. The same result follows for discounted reward SGs - as we saw in Section 1.1, discounting rewards is equivalent having some constant probability of the decision process being interrupted. Since NE all have the same value in zero-sum SGs, this means that we need only consider stationary strategies when looking for optimal NE strategies. Shapley (1953) proves the existence of the value vector and characterises it as the solution to an analogue of the Bellman equations we saw in Section 2.1. Assuming we start in state S and know how to play optimally from time $t+1$ onward, the optimal strategy at time t is the one solving the zero-sum matrix game $\Gamma(S, v_{t+1})$ with the following rewards:

$$\Gamma(S, v_{t+1}) = \left[R(S, A^1, A^2) + \gamma \sum_{S' \in \mathcal{S}} \mathbb{P}\left(S' \mid S, A^1, A^2\right) v_{t+1}(S') \right]_{A^1 \in \mathcal{A}^1(S), A^2 \in \mathcal{A}^2(S)}. \tag{10}$$

The value $v_t(S)$ of starting the game in state S at time t is therefore the same as the value of the matrix game $\Gamma(S, v_{t+1})$. Shapley's theorem states that the value vector exists and is the unique solution to the equations $v(S) = val(\Gamma(S, v))\ \forall S \in \mathcal{S}$, and the strategies for the game $\Gamma(S, v)$ are the optimal stationary strategies in state S of the stochastic game for both players. This theorem can be extended to show that stationary Nash equilibrium strategies also always exist in n-player discounted general-sum stochastic games (Raghavan (2003)).

Recall that we saw in Section 2.2 that the Bellman equations (since they characterise the value vector as a fixed point) can be used as an iterative update rule to find optimal stationary strategies. Shapley's theorem can also be used to define a sequence of approximations to the value function, by substituting the update $v^{(n)}(S) = val(\Gamma(S, v^{(n-1)}))$ into the value iteration algorithm (Algorithm 1).

4.3 Undiscounted Rewards

One complication of the stochastic game setting is that optimal stationary strategies do not always exist in the undiscounted case. A famous example of such a stochastic game is **the big match** (Filar and Vrieze (1997)). This is a zero-sum SG with 3 states with reward matrices:

$$\Gamma(1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \Gamma(2) = \begin{pmatrix} 0 \end{pmatrix}, \quad \Gamma(3) = \begin{pmatrix} 1 \end{pmatrix}.$$

States 2 and 3 are absorbing and both offer only one action to both players. In state 1, both players have 2 actions: if player 1 chooses action 1, the process stays in state 1. If player 1 chooses action 2, the process transitions to state 2 if player 2 chooses action 1 and state 3 otherwise. Clearly, $v(2) = 0$ and $v(3) = 1$, with the interesting question being how to play in state 1. In this SG, if player 1 uses a stationary strategy in state 1 then player 2 can always guarantee a limiting average payoff of 0. If player 1 chooses action 2 in state 1 with stationary probability $p > 0$, then if player 2 always chooses action 1 the process will transition to state 2 eventually with probability 1. If player 1 always chooses option 1, then player 2 can guarantee a payoff of 0 by always choosing action 2. Conversely, if player 2 uses a stationary strategy then player 1 can guarantee an expected limiting average payoff of at least 1/2. This means there can be no stationary NE strategies, since in either case a better limiting average payoff can be obtained for one player by switching to a different stationary strategy. However, it is possible to construct a non-stationary strategy for player 1 that gives an expected limiting average payoff arbitrarily close to 1/2 in state 1, by choosing action 2 with a probability that depends on the number of times player 2 has been observed to take action 2 (Raghavan and Filar (1991)).

As we will see in the next section, stationary strategies do still exist for many classes of undiscounted SGs, since in particular the ordered field property implies the existence of an optimal stationary strategy. Minimal and sufficient conditions for their existence are also known (Raghavan and Filar (1991)).

4.4 The Ordered Field Property

An additional difficulty with stochastic games is that there is no guarantee of being able to find the optimal stationary policy in a finite number of iterations (Raghavan and Filar (1991)). For example, it is possible for a SG with rational rewards, transition probabilities, and discount factor to have an optimal strategy with irrational entries, so that it is not possible to compute the true optimal strategy from the game data in a finite number of arithmetic operations. In particular, formulation as a linear program is not possible for games like this, since LPs can be solved in finitely many iterations using the simplex method.

This issue arises for stochastic games without the **ordered field property**. This can be defined as follows (Raghavan (2003)): a SG has this property if it has an optimal stationary solution whose entries lie in the smallest ordered subfield of $\mathbb{R}$ containing the data for the game (i.e. the rewards, transition probabilities, and discount factor), so that the entries of that solution can be generated from the data by finitely many arithmetic operations. The ordered field property has been proven for the following classes of discounted and undiscounted SGs (Raghavan (2003)):

- **Perfect information** games: in each state, at most one player as more than one action.
- **Single controller** games: only one player's actions can affect the transition probabilities.
- **SER-SIT** (SEparable Reward - State Independent Transition) games: the rewards can be written as $R(S, A^1, A^2) \equiv R(S) + R_A(A^1, A^2)$ and the state transition probabilities do not depend on S.
- **Switching controller** zero-sum games: in each state, the transition probabilities depend on at most one player's actions.
- **ARAT** (Additive Reward, Additive Transition) zero-sum games: the rewards satisfy $R(S, A^1, A^2) \equiv R^1(S, A^1) + R^2(S, A^2)$ and the transition probabilities satisfy $\mathbb{P}\left(S' \mid S, A^1, A^2\right) \equiv P^1(S' \mid S, A^1) + P^2(S' \mid S, A^2)$.

Finite-step algorithms have been found for some of the classes identified above, although this is still an open problem for others (Raghavan (2003)). For example, linear programming formulations exist for single player controlled games (both discounted and undiscounted). Note that MDPs and matrix games always possess the ordered field property, since they can be solved with linear programming.

Complete characterisation of the class of games for which this property holds in $\mathbb{Q}$ was identified as an open problem by Raghavan and Filar (1991). Avrachenkov et al. (2019) proved that it holds for all zero-sum discounted and undiscounted games in the field of algebraic numbers, which means that for all rational SGs we can at least guarantee that the value can be found as a root of a suitably constructed polynomial. Avrachenkov et al. (2019) note, however, that while such polynomials can be constructed in a finite number of steps the same does not hold true for their roots, which cannot in general be found in closed form or by finite algorithms.

4.5 Patrolling Problem Examples

The patrol problem described in Section 1.3 lends itself to being seen as a stochastic game, since in real life attackers are likely to be strategic, or it may be necessary to coordinate multiple patrollers. Lin et al. (2013) considered a reformulation of their examples with strategic attackers who arrive at random and do not know the location of the patroller on when they arrive. In this case, the patroller plays a zero-sum game with random rewards against potential attackers. This can be considered as a SG with a single state, with the patroller choosing a patrol route and the attackers a node simultaneously. They solved this version exactly with linear programming, and approximately using an index heuristic strategy based on that for the MDP case.

One example of a sequential SG formulation of the patrol problem is the **travelling inspector** problem (Filar and Schultz (1986)). In this framework, the patroller is an inspector who aims to detect violations of regulations by their inspectees. Here the inspector must decide which location to visit next, while the inspectees may coordinate to decide where a violation should occur given the inspector's current location. An example is the midnight dumping problem, where the inspector visits locations where toxic waste might be dumped, seeking to minimize the long-run average rate waste is dumped undetected. Here it is assumed that the violation is completed at a random time within the time period, and the inspector arrives at their next location after some fixed travel time depending on the start and end locations, detecting a violation only if it occurs while they are present. The inspectees coordinate to decide on the fraction of waste to send to each location. This is an undiscounted zero-sum single controller stochastic game (since the state can be considered to be the inspector's current location), and was solved by Filar and Schultz (1986) using linear programming.

Further extensions are possible by considering different information structures - the two examples here assume that the patroller's location is either completely known or unknown to the attackers, which is necessary for SG formulation since it is assumed that all players are aware of the current state. Extensions that involve partial observation of the system state include the case where attackers can only observe whether or not a patroller is present at their current location before deciding when to begin an attack (which requires a fixed number of uninterrupted time periods to complete), known as the **uniformed patroller** case. This example was formulated and solved for some network topologies by Alpern and Katsikas (2019). Note that this could be seen as a competitive version of the MDP formulation of Lin et al. (2013), in which attackers partially observe the state (the patroller's current location) and may choose to either wait or begin their attack at each timestep. This would be an example of a **partially observable stochastic game** (POSG), an extension which we will discuss in Section 5.2. POSGs and POMDPs (partially observable MDPs) are much harder to solve in general than SGs and MDPs, since it is usually necessary to base decisions on the history of observations rather than just the current state - for example, in the solution of Alpern and Katsikas (2019) the attacker must keep track of the number of time periods since the patroller was last seen.

5 Extensions and Open Research Areas

In Section 3 we focused on tabular RL algorithms. This approach scales poorly to large state-action spaces, since the task of storing and estimating a large number of Q-values can be computationally infeasible. Most current research in reinforcement learning handles this by modelling Q-values approximately by a parametrised function, often a deep neural network. This works very well empirically, and can for example allow for approximate handling of partial observability (discussed in more detail in Section 5.2) by incorporating information from past observations as the hidden state in a recurrent neural network model for Q-values (Dulac-Arnold et al. (2019)). However, relatively little is known about the convergence properties of deep learning methods in this context (Zhang et al. (2021)).

Many theoretical questions about stochastic games remain open - for example, complete characterisation of the games in $\mathbb{Q}$ with the ordered field property. Avrachenkov et al. (2019) recently proved the property for zero-sum SGs in the algebraic numbers, and gave a method for checking if a rational SG will have a rational solution. In the general-sum setting, it is not clear how to play optimally even if all NE can be found, as NE can have different values (Peterson (2017)). Since non-stationary Nash equilibria can have better payoffs for both players (e.g. the iterated prisoners' dilemma mentioned in Section 4.1), looking exclusively for stationary equilibria may not be the "correct" aim at all in this setting.

5.1 Multi-Agent Reinforcement Learning

Section 4 focused on introducing the theory of stochastic games and on generalising the ideas of Section 2 to the stochastic game setting. An active field of research is in applying reinforcement learning techniques to the problem - this field is also known as **multi-agent reinforcement learning** (MARL) (Zhang et al. (2021)). This introduces several additional difficulties to the reinforcement learning problem - for example, interaction with the system also necessitates interaction with other players, so that multiple agents learning simultaneously will face a non-stationary environment as other agents adapt their behaviour. In real-world settings it may also be necessary to guard against adversarial opponents who actively seek to disrupt learning.

Several variations of Q-learning have been developed for the multi-agent setting, which perform well in zero-sum and co-operative games, as long as it is possible to guarantee that each state-action pair is visited infinitely often. Such methods face convergence issues in the general-sum setting - convergence guarantees are only possible under strict assumptions on the structure of the SG (Zhang et al. (2021)). Zinkevich et al. (2005) showed that methods based on estimating Q-values will fail to converge to a stationary equilibrium on a whole class of general-sum SGs, in which Q-values are insufficient for deriving all stationary equilibria. They suggested an alternative notion of equilibrium for general-sum games: **cyclic equililbrium**, a non-stationary NE in which agents cycle between a fixed set of stationary policies, and showed that value-based methods do typically succeed in finding them.

5.2 Partial Observability

In Section 4.5 we saw an extension to the patrol problem with strategic attackers where attackers could observe whether the patroller is present at their current location or not, which can be modelled as access to partial observations of the system state. This is a sensible extension for many real life decision problems, where data about the system may be noisy or incomplete, or differ in unknown ways from the simulated environment used to learn policies (Dulac-Arnold et al. (2019)). Similarly, in problems we might want to model as stochastic games, players may not all have access to the same information - for example, in card games each player can typically only see their own hand so can only access part of the full system state.

In these cases, optimal policies can depend on past observations, since it is possible to use them to deduce information about the current state. In **partially observable MDP**s (POMDPs) it is necessary to maintain a **belief state**, a posterior distribution over all states given previous observations (Sutton and Barto (2018)). The situation for **partially observable stochastic games** (POSGs) is considerably more complicated since players must form beliefs over not only possible states but also the policies and beliefs of their opponents, making the problem highly computationally intractable in general (Hansen et al. (2004)).

Most current research on POSGs focuses on solution methods for particular classes that are known to be more tractable - for example: co-operative games, games where only one player has imperfect observations, and games where player observations are public (Horák and Bošanský (2019)). Much work has been done on the co-operative case where all agents have a common reward function, also known as a **decentralised POMDP** (Dec-POMDP) (Zhang et al. (2021)). There has been very little work done on solving POSGs in general - one example is Hansen et al. (2004), who developed an exact dynamic programming algorithm for general POSGs with a finite time horizon. Their method was based on combining DP methods for POMDPs with dominated strategy elimination, which they showed yielded the optimal solution in the Dec-POMDP setting.

References

Alpern, S. and Katsikas, S. The uniformed patroller game. *arXiv preprint 1908.01859*, 2019.

Avrachenkov, K., Ejov, V., Filar, J. A., and Moghaddam, A. Zero-sum stochastic games over the field of real algebraic numbers. *Dynamic Games and Applications*, 9:1026–1041, 2019.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. Challenges of real-world reinforcement learning. *arXiv preprint 1904.12901*, 2019.

Filar, J. and Vrieze, K. *Competitive Markov Decision Processes*. Springer, New York, NY, 1997.

Filar, J. A. and Schultz, T. A. The traveling inspector model. *OR Spectr.*, 8(1):33–36, 1986.

Gosavi, A. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. Dynamic programming for partially observable stochastic games. AAAI'04, page 709–715. AAAI Press, 2004.

Horák, Ka. and Bošanský, B. Solving partially observable stochastic games with public observations. AAAI'19/IAAI'19/EAAI'19, page 2029–2036. AAAI Press, 2019.

Lin, K. Y., Atkinson, M. P., Chung, T. H., and Glazebrook, K. D. A graph patrol problem with random attack times. *Operations Research*, 61(3):694–710, 2013.

Lin, K. Y., Atkinson, M. P., and Glazebrook, K. D. Optimal patrol to uncover threats in time when detection is imperfect. *Naval Research Logistics (NRL)*, 61(8):557–576, 2014.

Peterson, M. *An Introduction to Decision Theory.* Cambridge Introductions to Philosophy. Cambridge University Press, 2nd edition, 2017.

Powell, W. B. *Approximate Dynamic Programming : Solving the Curses of Dimensionality.* Wiley series in probability and statistics. J. Wiley Sons, Hoboken, N.J., 2nd edition, 2011.

Raghavan, T. E. S. Finite-step algorithms for single-controller and perfect information stochastic games. In *Stochastic games and applications (Stony Brook, NY,1999)*, volume 570 of *NATO Sci. Ser. C Math. Phys. Sci.*, pages 227–251. Kluwer Acad. Publ., Dordrecht, 2003.

Raghavan, T. E. S. and Filar, J. A. Algorithms for stochastic games—a survey. *Z. Oper. Res.*, 35(6):437–472, 1991.

Ross, S. M. *Introduction to Stochastic Dynamic Programming.* Probability and mathematical statistics. Academic Press, London, 1995.

Shapley, L. S. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10): 1095–1100, 1953.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning : An Introduction.* Adaptive computation and machine learning series. The MIT Press, Cambridge, MA, 2nd edition, 2018.

Watkins, C. J. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, 1989.

Watkins, C. J. and Dayan, P. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

White, D.J. Dynamic programming, Markov chains, and the method of successive approximations. *Journal of Mathematical Analysis and Applications*, 6(3):373–376, 1963.

Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. In *Handbook of Reinforcement Learning and Control*, pages 321–384. Springer International Publishing, Cham, 2021.

Zinkevich, M., Greenwald, A., and Littman, M. Cyclic equilibria in Markov games. In *Advances in Neural Information Processing Systems*, volume 18, page 1641–1648. MIT Press, 2005.