Markov Decision Processes *

1 MDPs

Decision making is an important task, especially temporally. Both the decision to be made and it's impact have both immediate (short-term) and long-term effects. Immediate effects are transparent and can be judged at the current time-step, but long term effects are not always as transparent. It is easy to think of examples where we can have good short-term effects that result in poor overall long-term effects. We therefore want to choose an action that makes the right tradeoff to yield the best possible overall solution to maximise our reward.

We use a Markov decision process (MDP) to model such problems to automate and optmise this process.

The components of an MDP model are:

- A set of states S: These states represent how the word exists at different time points. Every such state i.e., every possible way that the world can be, is a state in the MDP.
- A set of actions A: These actions can affect the current state of the world, resulting in a new state; thus resulting in some effect and a different set of further actions that can be taken. An important point of MDPs is that effects of an action are probabilistic. Therefore, an action for a particular state can result in different states with different probabilities, thus allowing us to make a statistical decision.
- Effects i.e., a transition function $T(s, a, s')$: These are a result of taking an action at a state, as described above. This is also called the model or the dynamics and defines the probability of reaching state s' from state s with action a i.e., $P(s'|s, a)$.
- Reward i.e., a reward function $R(s, a, s')$: These are a defined value for a $(state, action)$ pair i.e., the immediate reward value we obtain by taking an action at a particular state.

*This covers MDPs, CO-MDPs, POMDPS and algorithms for solving. Code is in a GitHub repo (link). These notes are taken from Michael Littman's exceptionally informative tutorial at Brown University; for more detail and lectures consult his homepage.

The solution to an MDP is called a *policy* i.e., a mapping that defines for each state, the best possible action to take. For the sake of simplicity, we limit to *finite horizon* solutions i.e., we assume that agents do not exist for infinite time periods. Therefore action values and rewards can decay with increasing time according to some γ, thus urging agents to take action sooner rather than later. We also compute a *value function* that is similar to a policy i.e., a mapping that defines for each state, a numerical value. From this value function, we can then easily derive the policy i.e., given a numerical *value* for each state we can proceed to specify an *action* for each state i.e., define a policy.

Note that our MDPs, as their names specifically state, follow the *Markov* assumption. This basically says that for an MDP, the next state is solely determine by the current state and action. Note that there exist situations when we want to look back further i.e., the effect of an action might not depend on the previous states, but can go back several states further. Our first-order Markovian assumption however, will not let us model these situations directly.

Given an MDP, our goal is to find a policy, which is easy if we have knowledge of the values of states for a horizon length. To do this let's introduce the **Value Iteration Algorithm** i.e., a way to compute the value function by finding a sequence of value functions, each derived from the previous one. Similar to every recursion function, we first define the base case. This is when our horizon length = 1 i.e., when we are at a state and need to make a single decision. Since we know the immediate rewards, for each state, we simply choose the action corresponding to the $argmax$ over all rewards. Now the second iteration deals with horizon length = 2 i.e., when we have two steps forward. Here we want the immediate reward for the action we take, plus the reward for the future action. We've already computed horizon length=1 for every state, so for each action, we simply sum the current reward plus the computed reward for the state it leads to and again, choose the action corresponding to the $argmax$ over all rewards. We continue in this way until convergence. In all, this involves iterating over all states and weighting the values with transition probabilities. The formula is given below:

$\mathrm{V}^*_{i+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*_i(s')]$(1)

This is also called the Bellman update. In short: $V^*_i(s)$ is the expected sum of rewards on starting from state s and acting optimally for horizon length=i.

How do we know this converges?

We want to show that $\lim_{k\to\infty} V_k = V^*$ i.e., we converge to unique optimal values. We know that for two approximations U and V, $|U_{i+1} - V_{i+1}| \leq \gamma\,|U_i - V_i|$ i.e., as we continue, approximations get closer to one another. Therefore any approximation must get closer to the most optimal U and this therefore con-

verges to a unique, stable, optimal solution.

So in all, *value iteration* allows us to repeatedly update an estimate of the optimal value function according to the Bellman optimality equation, and we can thus compute the optimal policy (read: optimal value function). This therefore involves iterating over all states and using the transition probabilities to weight the values.

Let's now look at **Policy Iteration**.

This manipulates the policy indirectly, rather than finding it through value iteration. We start with some initial policy π_o and at each iteration, perform a *policy evaluation* and a *policy improvement*. This therefore includes computing the policy at each iteration, solving the linear equation and then improving it, until we find the optimal $\pi^{'}$.

The convergence property of policy iteration: $\pi \rightarrow \pi^*$.

We can easily prove this by showing that each iteration is a contraction; therefore the policy must either improve at each step, or it must already be the optimal policy. Since the number of policies is finite (albeit exponentially large), policy iteration converges to the *exact* optimal policy. Note that in theory, we could take an exponential number of iterations before we converge, but for most problems of interest, convergence occurs much faster. Interestingly, policy iteration requires fewer iterations than value iteration and gives us the *exact* value function. Value iteration often converges to the optimal policy long before the approximation of the value function is the correct value, but this is MDP-specific.

Do we want to use Policy Iteration or Value Iteration?

From above, we know that policy iteration requires fewer iterations than value iteration. However each iteration requires solving a linear system; which is in contrast to value iteration, where each iteration only requires applying the Bellman operator.

In practice, policy iteration is often faster, especially when the transition probabilities are structured and sparse, which makes solving the linear system far more efficient.

It is interesting to note that a method that combines ideas from both, called *modified policy iteration* (Putterman and Shin, 1978) solves the linear system approximately, using backups similar to value iteration. This often performs better than both policy iteration or value iteration.

2 POMDPs

https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-background.html

We can think of POMDPs as the MDPs and CO-MDPs i.e., regular discrete Markov decision processes, but with partial observability. This is far from a trivial addition, considering that all solution procedures for regular MDPs give values or policies for states, but require the state to be completely known at all times.

Like a regular MDP, a POMDP has a set of states, a set of actions, transition weights, immediate rewards and action effects on states. The difference is whether or not we can observe the current state of the process. Instead of directly observing the current state, we get an *observation* that gives us a hint about the state that it is in. These observations can be probabilistic, so we need to specify the observation model i.e., the model that tells us the probability of each observation for each state in the model.

The underlying dynamics of the POMDP are Markovian, but we have no direct access to the current state, which therefore means our decisions require us to keep track of the history of the process. This could possibly be the entire history, making this a non-Markovian process.

However, maintaining a probability distribution over all the states, gives us the same information as maintaining the complete history. We therefore maintain a distribution over states, and update this when performing an action and making an observation. These updates involve using the transition and observation probabilities.

https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-solving.html

2.1 Solving POMDPs

In regular MDPs, the problem that we want to solve is to find a mapping from states to actions; however in POMDPs our problem is to find a mapping from a probability distribution over states to actions. This probability distribution over states is a *belief state* and the entire probability space i.e., the set of all possible probability distributions is the *belief space*.

For example, in a two-state POMDP, our belief state can be represented by a single number that represents the probability of being in the first state, which allows us to simply infer the probability of the only other state i.e., the second state. Therefore, this entire space of belief states can be represented as a line segment albeit of significant width; which will clarify later explanations. These lines become hyperplanes for higher dimensional examples.

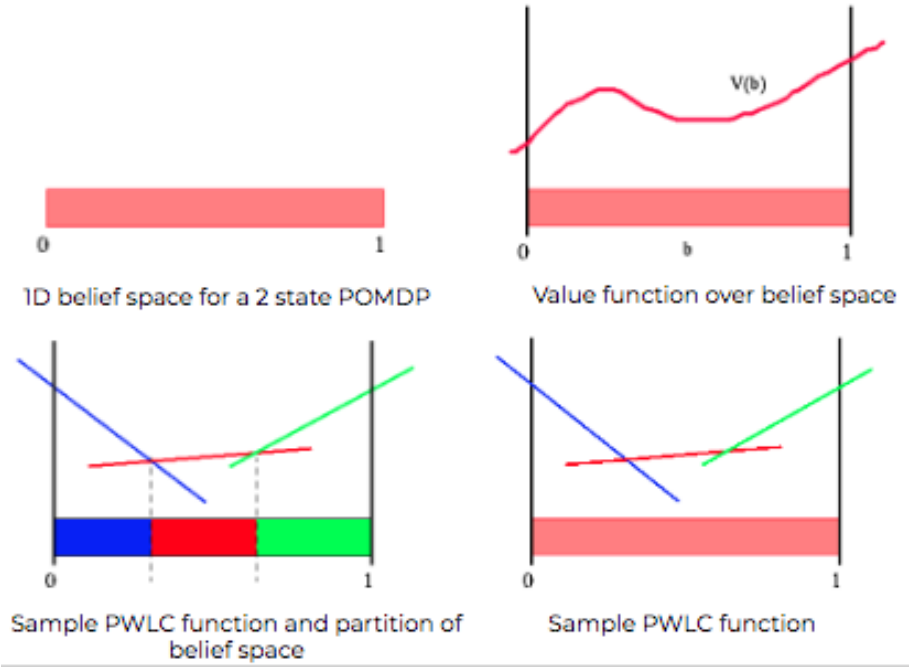
The belief space is therefore a single line segment, labeled with a 0 and 1 on

either ends that tells us the probability of being in state s_1.

Consider the problem of updating belief state, assuming we start at a particular belief state b. If we take action a_1 and receive observation z_1 by taking the action, then our next belief state is fully determined. Since we assume that there are a finite number of actions and observations; given a belief state, there are a finite number of possible next belief states, each corresponding to a combination of action and observation. Consider the figure, where the red line corresponds to our 1D belief space for a 2 state POMDP, the yellow dot is the initial belief state, and the arcs represent the process of transforming the belief state. We can therefore see that with a finite number of arcs (actions that correspond to observations), we have a finite number of transformations, thus a finite number of possible belief states.

Given that our observations are probabilistic, each belief state has a probability associated with it. Before we take an action, each resulting belief state has a probability associated with it, and there are multiple possible next belief states (the number of observations for a given action). Keep in mind that for a given action, the next belief state probabilities must sum to 1. It turns out that the process of maintaining a belief state is Markovian; the next belief state depends only on the current belief state (and therefore current action and observation). Therefore we can convert a discrete POMDP problem into a continuous space CO-MDP problem, where the continuous space is the belief space. The transitions of this new continuous space CO-MDP can be derived from the transition and observation probabilities of the POMDP, and we are tehrefore back to solving a CO-MDP and can use an adapted version of the value iteration algorithm.

We do however have to consider the continuous state space which is different from our CO-MDP. Earlier, in value iteration, we could mainting a table with one entry per state, and the value of each state could be stored in the table, giving us a finite representation of the value function. For our POMDP, however, we have a continuous space and the value function is some arbitrary function over the belief space. Our first problem is therefore, to represent this value function. The POMDP formulation imposes restrictions on the form of the solutions to the continuous space CO-MDP derived from the POMDP. The key insight is that the finite horizon value function is *piecewise linear and convex* (PWLC) for every horizon length. This essentially means that for each iteration of value iteration, we need to only find a number of linear segments (that is finite) that make up the value function. These linear segments comepletely specify the value function over the belief space. In higher dimensions these are not linear, but hyperplanes through our belief space, and we can represent each hyperplane with a vector of numbers i.e., the coefficients of the hyperplane equation. The value at any belief state is then found by plugging in the belief state into the hyperplane equation (or more simply, if both the hyperplane and belief state are vectors, the value of a belief point is the dot product of the two).



Therefore, the value function for each horizon is represented for each horizon, as a set of vectors, and what we want i.e., the value of a belief state, is the vector that has the largest dot product with the belief state. Note that instead of linear segments over our belief space, another way to view the function is as something that partitions the belief space into a finite number of segments. Consider the figures below.

Now for our value iteration algorithm, consider the continuous space CO-MDP that we want to adapt value iteration to. From above, since each horizon's value function is PWLC, we can represent the value function at each iteration as a set of vectors i.e., coefficients of the hyper-planes. But now, in each iteration of value iteration in the discrete state space, we can find a state's new value by looping over all possible next states. But when in a continuous state CO-MDP, we cannot enumerate all possible states. There are therefore specific algorithms that try to solve this difficulty i.e., given a value function as a set of vectors for horizon h, we want to generate the set of vectors for the value function of horizon $h + 1$.

https://cs.brown.edu/research/ai/pomdp/tutorial/pomdp-vi-example.html