5. Online Planning - Part I.

In this lecture we

1. introduce online planning;
2. show that for deterministic MDPs there is an online planner whose runtime per call is independent of the size of the state space;
3. show that this online planner has in fact a near-optimal runtime in a worst-case sense.

What is Online Planning?

In a previous lecture we have seen that in discounted MDP with S states and A actions, no algorithm can output a $\delta \leq \gamma/(1-\gamma)$ optimal or better policy with a computation cost less than $\Omega(S^2A)$ provided that the MDP is given with a table representation. One of the SA factors here comes from that to specify a policy one needs to compute (and output) what action to take in every state. The additional S factor comes from because to figure out whether an action is any good, one needs to read almost all entries of the next-state distribution vector.

An unpleasant tendency of the world is that if a problem is modelled as an MDP (that is, the Markov assumption is faithfully observed), the size of the state space tends to blow up. Bellman's curse of dimensionality is one reason why this happens. To be able to deal with such large MDPs, we expect our algorithm's **runtime to be independent of the size of the state space**. However, our lower bound tells us that this is a pipe dream.

But why did we require the planner to output a full policy? And why did we assume that the only way to get information about the MDP is to read big tables of transition probabilities? In fact, if the planner is used inside an "agent" that is embedded in an environment, there is no need for the planner to output a full policy: In every moment, the planner just needs to calculate the action to be taken in the state corresponding to the current circumstances of the environment. In particular, there is no need to specify what action to take under any other circumstances than the current one!

As we usually do in these lectures, assume that the environment is an MDP and the agent gets access to the state in every step when it needs to make a decision. Further, assume that the agent is lucky to also have access to a simulator of the MDP that describes its environment. Just think of the simulator as a black box that can be, fed with a state-action pair and responds with the immediate reward and a random next state from the correct next-state distribution. One can then perhaps build a planner that uses this black box with a "few" queries and quickly returns an action, to be taken by the agent, moving the environment to a random next state, from where the process continues.

Now, the planner does not need to output actions at all states and it does not need to spend time on reading long probability vectors. Hence, in theory, the obstacles that led to the lower bound are removed. The question still remains whether in this new situation planner's can indeed get away with runtime independent of the size of the state space. To break the suspense, the answer is yes and it comes very easily for deterministic environments. For stochastic environments a little more work will be necessary.

In the remainder of this lecture we give a formal problem definition for the **online planning problem** that was described informally above. Next, the result is explained for deterministic environments. This result will be matched with a lower bound.

Online Planning: Formal Definitions

We start with the definition of MDP simulators. We use a language similar to that used to describe optimization problems where one talks about optimization in the presence of various oracles (zeroth-order, first order, noisy, etc.). Because we assume that all MDPs are finite, we identify the state and action spaces with subsets of the natural numbers and for the action set we also require that the action set is $[\mathrm{A}]$ where A is the number of actions. This simplifies the description quite a bit.

Definition (MDP simulator): A simulator implementing an MDP $M = (\mathcal{S}, \mathcal{A}, P, r)$ is a "black-box oracle" that when **queried** with a state action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ returns the reward $r_a(s)$ and a random state $S' \sim P_a(s)$, where $r = (r_a(s))_{s,a}$ and $P = (P_a(s))_{s,a}$.

Users of the black-box must pay attention avoid querying it for state-action pairs outside of $\mathcal{S} \times \mathcal{A}$. Our next notion is that of an online planner:

Definition (Online Planner): An online planner takes as input the number of actions A, a state $s \in \mathbb{N}$, an MDP simulator "access point". After querying this simulator finitely many times, the planner needs to return an action from $[\mathrm{A}]$.

(Online) planners may randomize their calculation. Even if they do not randomize, the action returned by a planner is in general random due to the randomness of the simulator that the planner uses. A planner is **well-formed** if no matter what MDP it interfaces with through a simulator, it returns an action after querying the simulator finitely many times. This also means that the planner can never feed the simulator with state-action pair outside of the set of such pairs.

If an online planner is given access to a simulator of M, the planner and the MDP M together induce a policy of the MDP. We will just refer to this policy as the planner-induced policy π when the MDP is clear from the context. Yet, this policy depends on the MDP implemented by the simulator. If an online planner is well-formed, this policy is well-defined no matter the MDP that is implemented by the simulator.

Online planners are expected to produce good policies:

Definition (δ-sound Online Planner): We say that an online planner is δ-sound if it is well-formed and for any MDP M, the policy π induced by it and a simulator implementing M is δ-optimal in M. In particular,

$$v^\pi \geq v^* - \delta \mathbf{1}$$

must hold where v^* is the optimal value function in M.

The (per-state, worst-case) **query-cost** of an online planner is the maximum number of queries it submits to the simulator where the maximum is over both the MDPs and the initial states.

The following vignette summarizes the problem of online planning:

Model:	Any finite MDP M

Oracle:	Black-box simulator of M
Local input:	State s
Local output:	Action A
Outcome:	Policy π
Postcondition:	$v_M^\pi \geq v_M^* - \delta \mathbf{1}$

As an optimization, we let online planners also take as input δ, the target suboptimality level.

Online Planning through Value Iteration and Action-value Functions

Recall value iteration:

1. Let $v_0 = \mathbf{0}$
2. For $k = 1, 2, \ldots$ let $v_{k+1} = T v_k$

As we have seen, if the iteration is stopped so that $k \geq H_{\gamma, \delta(1-\gamma)/(2\gamma)}$, the policy π_k defined via

$$\pi_k(s) = \arg\max_a r_a(s) + \gamma \langle P_a(s), v_k \rangle$$

is guaranteed to be δ-optimal. Can this be used for online planning? As we shall see, in a way, yes. But before showing this, it will be worthwhile to introduce some additional notation that, in the short term, will save us some writing. More importantly, the new notation will also be seen to influence algorithm design.

The observation is that to decide about what action to take, we need to calculate the one-step lookahead value of the various actions. Rather than doing this in a separate step as shown above, we could have as well chosen to keep track of these lookahead values throughout the whole procedure. Indeed, define $\tilde{T} : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ as

$$\tilde{T} q = r + \gamma P M q, \qquad (q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}),$$

where $r \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ and the operators $P : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$ and $M : \mathbb{R}^{\mathcal{S} \times \mathcal{A}} \to \mathbb{R}^{\mathcal{S}}$ are defined via

$$r(s,a) = r_a(s), \quad (Pv)(s,a) = \langle P_a(s), v \rangle, \quad (Mq)(s) = \max_{a \in \mathcal{A}} q(s,a)$$

with $s \in \mathcal{S}$, $a \in \mathcal{A}$, $v \in \mathbb{R}^{\mathcal{S}}$, $q \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$.

Then the definition of π_k can be shortened to

$$\pi_k(s) = \arg\max_a (\tilde{T}^{k+1}\mathbf{0})(s,a)\,.$$

It is instructive to write the above computation in a recursive, algorithmic form. Let

$$q_k = \tilde{T}^k \mathbf{0}.$$

Using a Python-like pseudocode, our function to calculate the values $q_k(s,\cdot)$ looks as follows:

```
1. define q(k,s):
2.  if k = 0 return [0 for a in A] # base case
3.  return [ r(s,a) + gamma * sum( [P(s,a,s') * max(q(k-1,s')) for s' in S] ) for a in A ]
4. end
```

Line 3, which is where the recursive call happens uses Python's list comprehensions: the brackets create lists and the function itself returns a list. This is a recursive function (since it calls itself in line 3. The runtime is easily seen to be $(\mathrm{AS})^k$, which is not very hopeful until we notice that if the MDP was deterministic, that is, $P(s,a,\cdot)$ has a single one entry, and we have a way of looking up which entry is this without going through all the states, say, $g : \mathcal{S}\times\mathcal{A}\to\mathcal{S}$ is a function that gives the next states, we can rewrite the above as

```
1. define q(k,s):
2.  if k = 0 return [0 for a in A] # base case
3.  return [ r(s,a) + gamma * max(q(k-1,g(s,a))) for a in A ]
4. end
```

As in line 3 there is no loop over the next states (no summing up over these), the runtime becomes

$$O(A^k)$$

which is the first time we see that a good action can be calculated with effort regardless of the size of the state space! And of course, if one is given a simulator of the underlying MDP, which is deterministic, calling g is the same as calling the simulator (once). But will this idea extend to the stochastic case? The answer is yes, but the details will be given in the next lecture. Instead, in this lecture we take a brief look at whether there is any possibility to do better than the above recursive procedure.

Lower Bound

Theorem (online planning lower bound): Take any online planner p that is δ-sound with $\delta < 1$ for discounted MDPs with rewards in $[0, 1]$. Then there exist some MDPs on which p uses at least $\Omega(\mathrm{A}^k)$ queries at some state with

$$k = \left\lceil \frac{\ln(1/(\delta(1-\gamma)))}{\ln(1/\gamma)} \right\rceil, \tag{1}$$

where A is the number of actions in the MDP.

Denote by k_γ the value defined in (1). Then, for $\gamma \to 1$, $k_\gamma = \Omega(H_{\gamma,\delta})$.

Proof: This is a typical needle-in-the-haystack argument. We saw in Question 5 on Homework 0 that no algorithm can find out which element of a binary array of length m is one with less than $\Omega(m)$ queries. Take a rooted regular A-ary tree of depth k. The tree has exactly A^k leafs. Consider an MDP with states corresponding to the nodes of this tree. Call the root s_0. Let the dynamics be deterministic: Taking an action at a node (of the tree) makes the next state the child of that node, unless the node is a leaf node, which are absorbing states: The next state under any action at any leaf state s is s itself. Let all the rewards be zero except at exactly one of the leaf nodes, where the reward under any action is set to one.

If a planner is δ-sound, we claim that it must find the optimal action at s_0. This holds because the value of this action is $\sum_{i=k}^{\infty} \gamma^i = \gamma^k/(1-\gamma)$ and, by our choice of k, $\gamma^k/(1-\gamma) \geq \delta$, while the value of any other action at s_0 is zero. It follows that the planner needs to be able to identify the unique action at the unique leaf node whose reward is one, which, by Question 5 on Homework 0, needs at least $\Omega(\mathrm{A}^k)$ queries. ■

Notes

Dealing with larger state spaces

For a fully formal specification the reader may worry about how a state is described to an online planner, especially, if we allowed uncountably many states. Because the online planner will only have access to the state that it receives as its input and the other states that are returned from the simulator, for the purpose of communication between the

online planner and its environment and the simulator, all these states can just be assigned unique numbers to identify them.

Gap between the lower and upper bound

There is an obvious gap between the lower and the upper bound that should be closed.

Local planning vs. online planning

Last year's lecture notes used the expression **local planning** in place of **online planning**. There are pros and cons for both expressions, but perhaps online planning better expresses that the planner will be used in an online fashion, that is, every time after a transition happens.

On simulators and access modes

Simulators come in many shapes and forms. A general planner needs to be prepared to be used in an interconnection with any simulator. But this is too much: Every simulator provides an interface to the planners and planners need to be designed around these interfaces. Therefore, planners will be specialized to the specific interface used. Here, we distinguish three types of interfaces based on what access the interface allows to generating data. The access can be **global**, **local** or **online**.

Global access means that the simulator provides a function that returns a description of the full state space. For finite MDPs this would just mean returning the number of states S. Then, the simulator can be called for any (s, a) pair where $s \in [S]$ and $a \in [A]$ (the simulator should also have a function that returns the number of actions, A). Internally, the simulator then needs to translate the integer indices s and a into appropriate data for which the simulation can be done. Then, the simulator would generate the next state, and translate it back to an integer in $[S]$, which is the data returned from the call. The simulator should also return the associated reward. Often, the reward would also be random (in the lecture, we are concerned with deterministic rewards, but this is just done for the sake of simplicity: random rewards at this stage would not create further difficulties).

Local access means that the simulator allows the planner to generate transitions starting only from states that were passed to the planner previously. To implement a local access simulator, one can just introduce an array that is used to remember all the states that have been returned to the planner. For the sake of interfacing with the planner, one can then use the indexing into this array. This way, the planner does not need to know the details of how states are internally represented and it also becomes possible to interface

with simulators where the number of states is infinite, or when it is finite, but calculating this number would be impractical or intractable. Of course, the simulator needs the ability to "go back" to a previously visited state and generate new transition data from there. This can be usually implemented on the top of existing simulators without much trouble (the ability to do this is known as "checkpointing").

Online access simulators have an "internal state", which the planners can manipulate in two ways: they can reset this internal state to the initial state (which is provided to the planner when the planner is called), or they can ask for a transition from the current internal state, by providing an action. As a result of this, the simulator's internal state would move to a random next state, which is what would be returned to the planner (along with the associated reward).

Clearly, any planner prepared to work with online access, can also be used with simulator that provide either local access or global access, and any planner prepared to work with local access can be used with simulators providing global access. In this way, online access is the most general of the access modes, local access is least general, and global access is the most restrictive.

Note that even with online access there is the issue that state information about the state of the environment has to be communicated to the planner in a way that is consistent with how state information can be passed from the planner to the simulator. To keep planners general, the environment and the simulator need to work on an appropriate consistent way of serializing information about the state, which is a pure engineering issue and can usually be done without much trouble.

"**Planning with a generative models**" is an alternative, early terminology that is still used in the literature today. Most commonly, this is means online planning with a global access simulator. However, as the expression itself is not as easy to adopt to different situations as described here, we will refrain from using it.
