Outline

- **Value Iteration**
- Asynchronous Dynamic Programming
- Generalized Policy Iteration
- Efficiency of Dynamic Programming
Second DP Method: Value Iteration

- A drawback of Policy Iteration - the need to perform policy evaluation in each iteration
  - Computationally heavy
  - Multiple sweeps through the state set
- Question: can we truncate the policy evaluation process?
  - Reduce the number of computations involved?
- Turns out we can - without losing convergence properties
- One such way is value iteration
  - Policy evaluation is stopped after just one sweep
  - Has the same convergence guarantees as policy iteration

\[
V_{k+1}(s) = \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V_k(s') \right]
\]
Value Iteration (cont.)

- Effectively embeds one sweep of policy evaluation and one sweep of policy improvement
- All variations converge to an optimal policy for discounted MDPs

Initialize $V$ arbitrarily, e.g., $V(s) = 0$, for all $s \in S^+$

Repeat

\[ \Delta \leftarrow 0 \]

For each $s \in S$:

\[ v \leftarrow V(s) \]

\[ V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]

\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that

\[ \pi(s) = \arg \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]
Example: Gambler’s Problem

- A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips.
  - If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake.
  - The game ends when the gambler wins by reaching his goal of $100, or loses by running out of money.

- On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars.

- This problem can be formulated as an undiscounted, finite (non-deterministic) MDP.
Gambler’s Problem: MDP Formulation

- The state is the gambler's capital $s = \{0, 1, 2, 3, ..., 100\}$
- The actions are stakes $a = \{1, 2, ..., \text{min}(s, 100-s)\}$
- The reward is zero on all transitions except those on which the gambler reaches his goal, when it is $+1$.
- The state-value function then gives the probability of winning from each state.
- A policy is a mapping from levels of capital to stakes
  - The optimal policy maximizes the probability of reaching the goal.
  - Let $p$ denote the probability of the coin coming up heads.
  - If $p$ is known, then the entire problem space is known and can be solved (i.e. complete model is provided).
Gambler’s Problem: Solutions for $p=0.4$
Asynchronous Dynamic Programming

- Major drawback of DP - computational complexity
  - Involve operations that sweep the entire state set
  - Example: Backgammon game has over $10^{20}$ states - even a million states per second would not suffice

- Asynchronous DP algorithms - in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set

- The values of some states may be backed up several times while the values of other states are backed up once
- The condition for convergence to the optimal policy - all states are backed up evenly in the long run
- Allow practical sub-optimal solutions to be found
Asynchronous Dynamic Programming

What we gain:

- Speed of convergence - no need for complete sweep every iteration
- Flexibility in selecting the states to be updated

It's a zero-sum game:

- For the optimal solution we need to perform all calculations involved in sweeping the entire state set
- Some states may not need to be updated as often as others
- Some states may be skipped altogether (as will be discussed later)

Facilitates real-time learning (learn and experience concurrently)

- For example, we can focus on the states the agent visits
Generalized Policy Iteration

- Policy iteration consists of two processes
  - Making the value function consistent with the current policy (policy evaluation)
  - Making the policy greedy with respect to the value function (policy improvement)

- So far we’ve considered methods that alternated between the two phases (on different granularities)

- Generalized Policy Iteration (GPI) refers to all variations of the above, regardless of granularity and details

- The two components can be viewed as both competing and cooperating
  - They complete in the sense of pulling in opposite directions
  - They complement as they lead to an optimal policy
Efficiency of Dynamic Programming

- DP may not be practical for large problems

- Compared to alternatives - DP is pretty good
  - Finding optimal policy is polynomial in states and actions
  - If $|A| = M$, and $|S| = N$ then DP is guaranteed to find the optimal policy in polynomial time
  - Even though total number of policies is $M^N$

- DP is often considered impractical because of the *curse of dimensionality*
  - $|S|$ grows exponentially with the number of state variables

- DP methods can solve (using PC) MDPs with millions of states

- On problems with large state spaces, asynchronous DP works best
  - Works because (practically speaking) only a few states occur along optimal solution trajectories