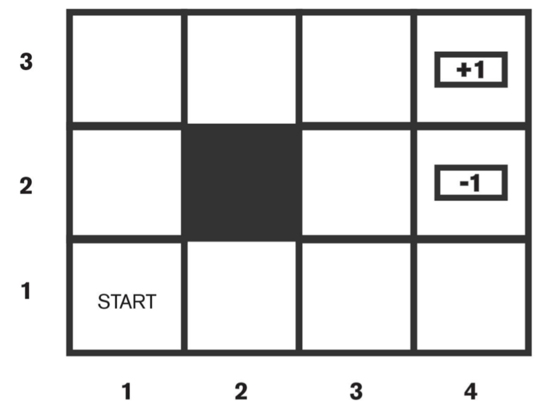
Sequential Decision-Making Under Uncertainty:
Solving MDPs

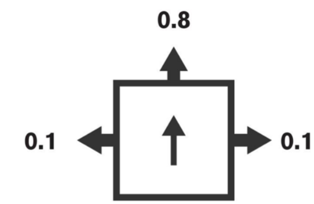
Siddharth Srivastava, Ph.D.
Assistant Professor
Arizona State University

Ira A. Fulton Schools of Engineering
Arizona State University

Recall: MDPs

- S **set of states**
  - E.g., At(1,1)





- A **set of actions**
- T **transition model**
  - $P(s'|s,a) = T(s,a,s')$
- $R: S \to \mathbb{R}$ **reward, or utility function**
  - Reward collected when timestep completes (agent is ready to act)
- **Agent can “drift”, end up in unintended states**
- **Solutions take the form of policies:** $\pi\colon S \to A$

Recall: Useful Equations for MDPs

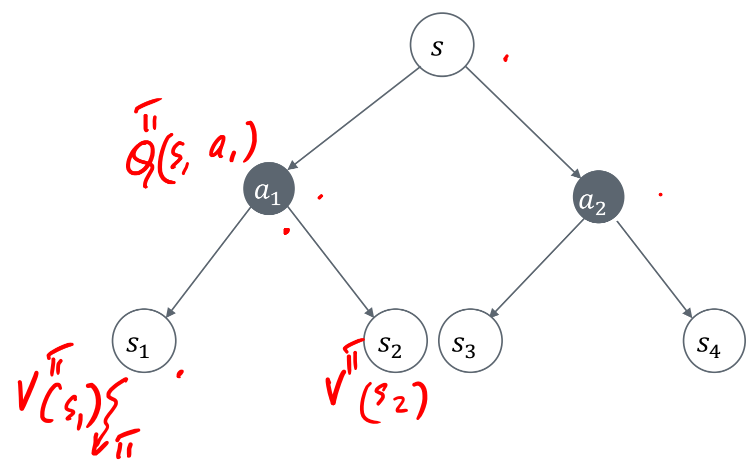
Value of a state s under a policy π

- $V^{\pi}(s)$ = expected utility starting in s and following π
- $Exp_{\pi}[\sum_t R(t)]$

Q-function: value of (s, a) pair

- $Q^{\pi}(s, a)$ = expected utility when executing a in s, then following π

  $= \sum_{s'} P(s'|s, a)\,[R(s) + \gamma V^{\pi}(s')]$



Optimal policy: π^*

- $\pi^*(s) = \mathrm{argmax}_{\pi} V^{\pi}(s)$

Convenient:

- Infinite horizon + discounting → π^* independent of time, starting state!

Computing Optimal Policies

Q function for the optimal policy:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\,[R(s) + \gamma V^*(s')]$$

V function for the optimal policy:

$$V^*(s) = \max_a Q^*(s,a)$$

Combining the two:

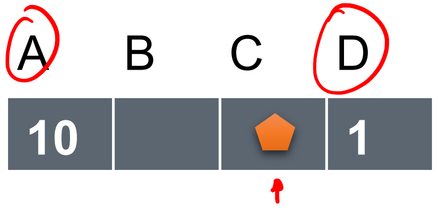
$$V^*(s) = \underbrace{\max_a \sum_{s'}}_{} P(s'|s,a)[R(s) + \gamma V^*(s')]$$

This is Bellman's Equation

Example

A, D: **terminal** states

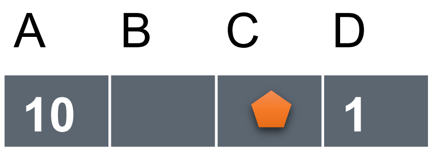
- Also called “trap” states
- Actions have no effects in them; zero reward after the first time they are reached



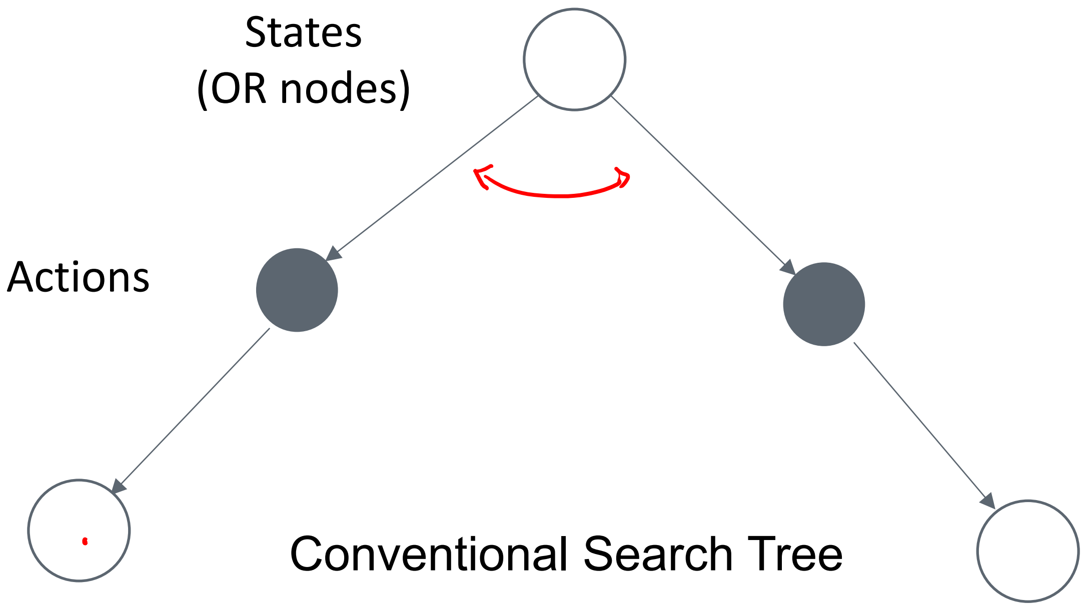
Transition probabilities:

- 0.8: move according to action executed,
- 0.2: stay;
- $\gamma = 0.9$

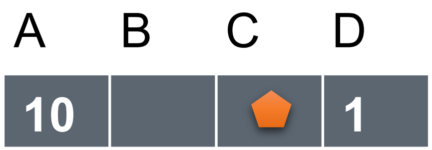
Solving MDPs: Stochastic Transitions



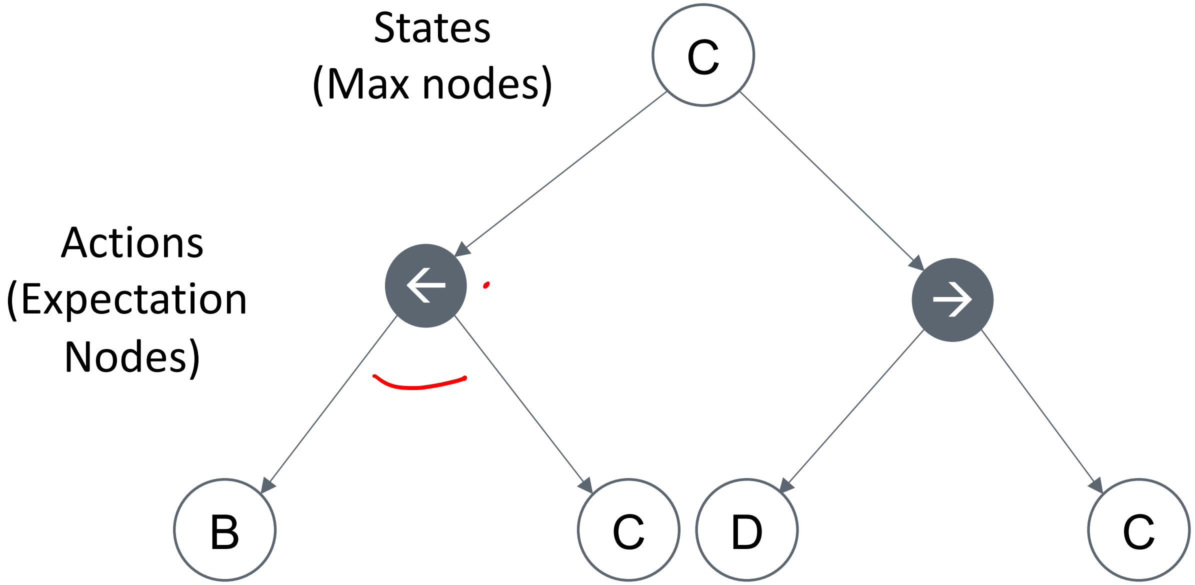
A, D: terminal states
Transition probability: 0.8 move according to action executed, 0.2 stay



Solving MDPs: Stochastic Transitions



A, D: terminal states
Transition probability: 0.8 move according to action executed, 0.2 stay



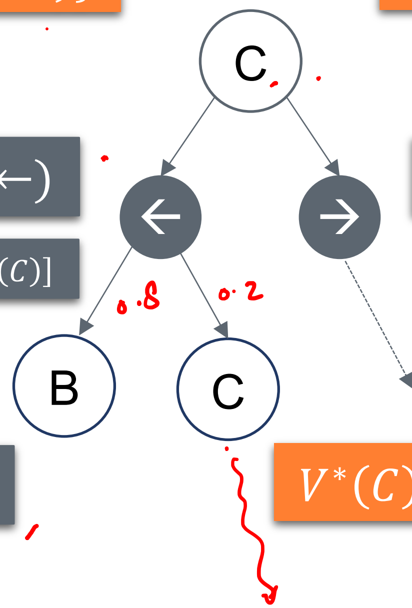
MDP Search Tree

How Would We Compute V^*?

$$V^*(s) = \max_a Q^*(s,a) = \max_a \sum_{s'} P(s'|s,a)\,[R(s) + \gamma V^*(s')]$$

$V^*(C) = \max\{Q^*(C,\leftarrow), Q^*(C,\rightarrow)\}$

$\pi^*(C) = \mathrm{argmax}_a\{Q^*(C,a)\}$



$Q^*(C,\leftarrow)$

$Q^*(C,\rightarrow)$

$= R(C) + \gamma[0.8V^*(B) + 0.2V^*(C)]$

$V^*(B)$

$V^*(C)$

A	B	C	D
10			1

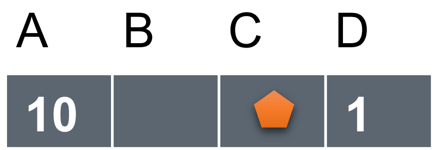
Problems:

- Bellman's equation is recursive
- Tree is of unbounded depth, repetitive

Solutions: dynamic programming, iterative computation

A, D: terminal states

Transition probability: 0.8 move according to action executed, 0.2 stay; $\gamma = 0.9$



	A	B	C	D
V_1	10	0	0	1

	A	B	C	D
V_0	0	0	0	0

$$V_1(s) = \max_a \sum_{s'} P(s'|s,a)[R(s\ )]$$

A, D: terminal states
Transition probability: 0.8 move according to action executed, 0.2 stay; $\gamma = 0.9$

A	B	C	D
10			1

$$
\begin{aligned}
&V_2(s) \\
&= max\{ \sum_{s'} P(s'|s, \leftarrow)[R(s\ ) + \gamma V_1(s')], \\
&\qquad \textstyle\sum_{s'} P(s'|s, \rightarrow)[R(s\ ) + \gamma V_1(s')]\}
\end{aligned}
$$

	A	B	C	D
V_2	10	7.2	.72	1

	A	B	C	D
V_1	10	0	0	1

	A	B	C	D
V_0	0	0	0	0

$$V_1(s) = \max_a \sum_{s'} P(s'|s,a)[R(s\ )]$$

A, D: terminal states
Transition probability: 0.8 move according to action executed, 0.2 stay; $\gamma = 0.9$

A	B	C	D
10			1

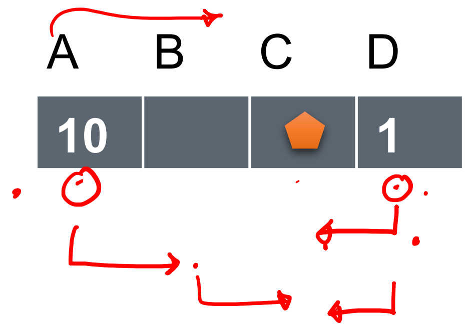
$$V_3(s) = max\{ \sum_{s'} P(s'|s,\leftarrow)[R(s\,) + \gamma V_2(s')], \sum_{s'} P(s'|s,\rightarrow)[R(s\,) + \gamma V_2(s')]\}$$

	A	B	C	D
V_2	10	7.2	.72	1

	A	B	C	D
V_1	10	0	0	1

	A	B	C	D
V_0	0	0	0	0

A, D: terminal states
Transition probability: 0.8 move according to action executed, 0.2 stay; $\gamma = 0.9$



	A	B	C	D
V_3	10	8.5	5.3	1

	A	B	C	D
V_2	10	7.2	.72	1

	A	B	C	D
V_1	10	0	0	1

	A	B	C	D
V_0	0	0	0	0

$$V_3(s) = max\{ \sum_{s'} P(s'|s,\leftarrow)[R(s) + \gamma V_2(s')], \sum_{s'} P(s'|s,\rightarrow)[R(s) + \gamma V_2(s')]\}$$

$V_i(s)$ gives the best possible expected total utility of starting from s, and executing i actions

"Value with i steps to go"

Value Iteration

- **The algorithm we just used is called value iteration**
  - Incrementally propagates the effects of R across the state space
- **Start with $V_0(s) = 0$ no time steps left** s
- $V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s\ ) + \gamma V_k(s')]$
- **Repeat until convergence**
  - (memoize $V_k(s)$ when it is first encountered)
- **When does this work?**
  - Theorem: Value iteration converges to the unique solution when $\gamma < 1$

Computing Actions from the Value Function

- Suppose we have the optimal values

- How should the agent act?

- We could use V* to compute best action (**policy extraction**):

  argmax $\{\leftarrow, \rightarrow\}$ $\{c, \rightarrow \quad c, \leftarrow\}$

  – $\pi^*(s) = argmax_a \sum_{s'} P(s'|s,a)[R(s) + \gamma V^*(s')]$

- More efficient: store Q*

$$\pi^*(s) = argmax_a Q^*(s,a)$$

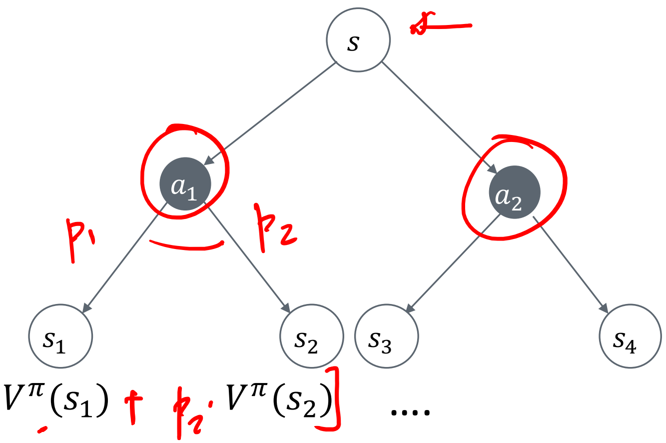
Limitations of Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s) + \gamma V_k(s')]$$

- $O(S^2A)$ **time per iteration**
- **Policy may have converged even when values haven't**
- **Policy iteration is another approach for computing V^* and π^* that addresses these issues**

Policy Iteration

- **Repeat steps until policy converges:**
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: lookahead one step (greedy) using converged (but not optimal!) values of subsequent states



- **Computes optimal policies**
- **Can converge (much) faster under some conditions**

Policy Iteration in Practice

- **Policy evaluation step: Compute a policy's value function using VI**
  - $V_{k+1}(s) = \max_{a} \sum_{s'} P(s'|s,a)\,[R(s) + \gamma V_k(s')]$
  - Does this make sense?

Policy Iteration: Details

Repeat until convergence:

- **Policy Evaluation**: First fix a policy, find its value function using VI

$$V_{k+1}(s) = \sum_{s'} P(s'|s, \pi_i(s))[R(s) + \gamma V_k^{\pi_i}(s')]$$

- This is easier than value iteration for computing the optimal V (why?)

- **Policy Improvement**: Then, improve the policy using a greedy update:

$$\pi_{i+1}(s) = argmax_a \sum_{s'} P(s'|s, a)[R(s) + \gamma V^{\pi_i}(s')]$$

This is looks ahead a single step using V^{π_i}

- Go back to policy evaluation for π_{i+1}.