Lecture 29

CS 6260 Automated Planning and Learning

Solving MDPs: Value Iteration

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Some material based on lecture slides for ASU's CSE571 Artificial Intelligence by Siddharth Srivastava.

Logistics

Logistics

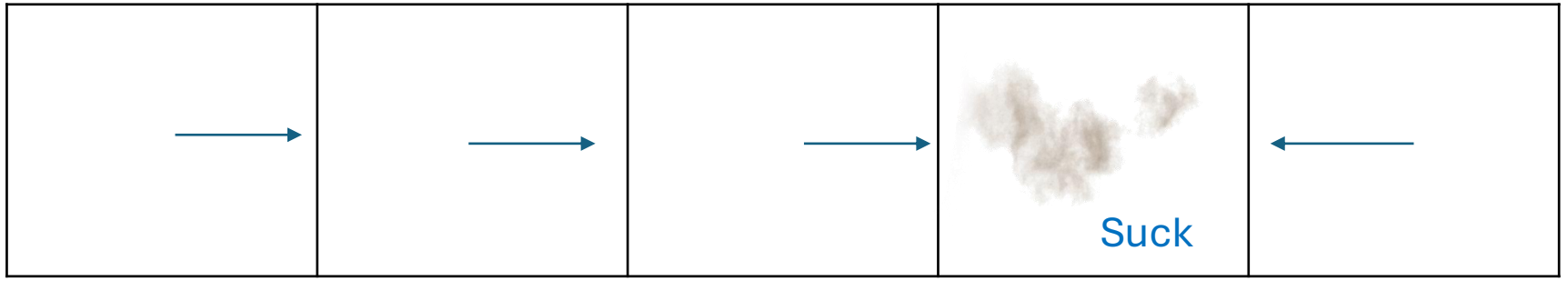
- Office Hours: 1 hour after class every week Tu, W, Th
- ROS based Assignment: Covered in Class on Tuesday Apr 21
  - https://wiki.ros.org/ROS/Tutorials
- Two more assignments
  1. Theory Assignment (like the last two)
  2. Programming Assignment

Recap: MDPs

Forms of Solutions

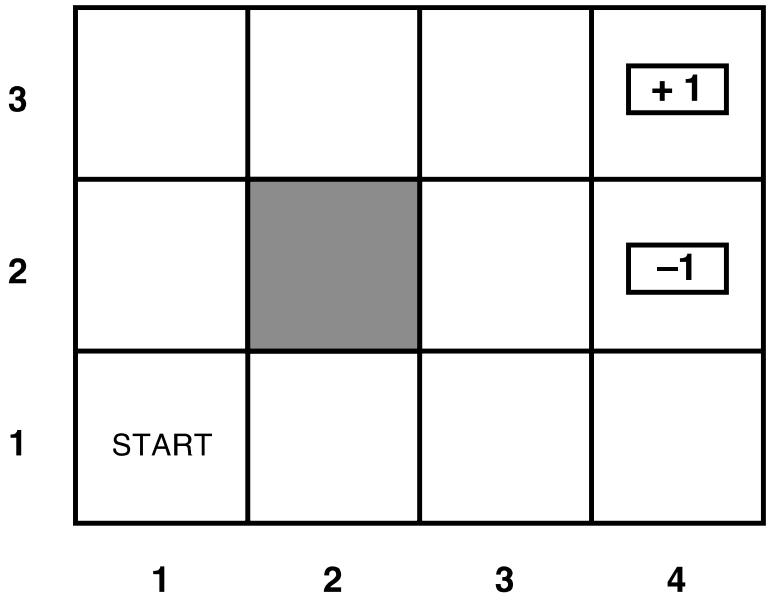
- Deterministic, fully observable
  - "Classical planning": solution is a plan, or a sequence of actions
- Deterministic, non-observable
  - "Conformant planning": solution is a plan, but often no solutions exist
- Non-deterministic, fully observable:
  - "Contingent planning": solution can be a policy (a function $S \rightarrow A$)

Non-determinism: a move action may fail

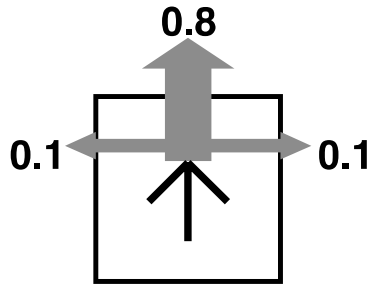


Markov Decision Processes

- S set of states
  - At(1,1)
- A set of actions
- P transition model
  - $P(s'|s,a)$
- $R: S \rightarrow \mathbb{R}$ reward, or utility function



(a)

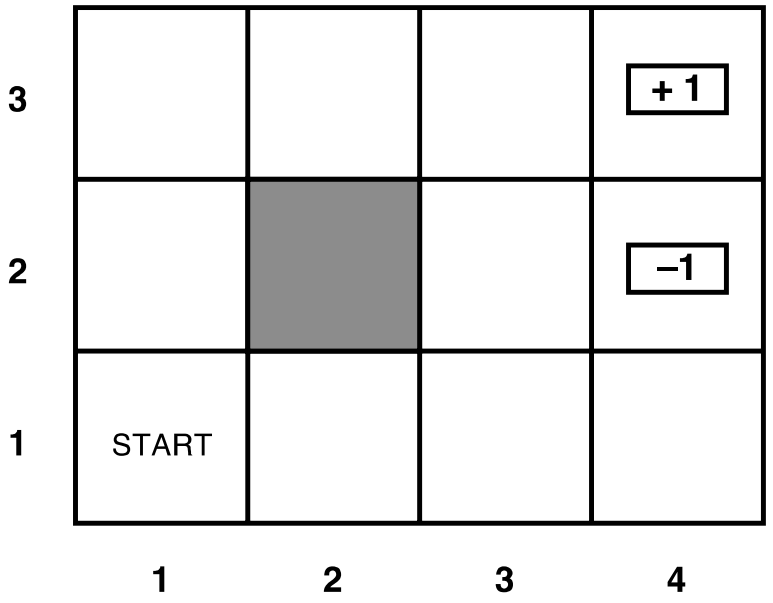


(b)

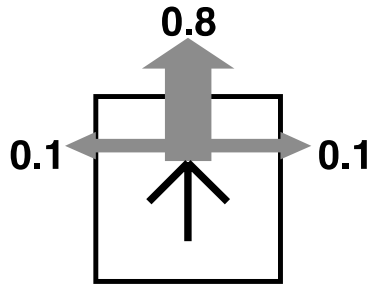
- Agent can "drift", end up in unintended states
- Solutions take the form of policies

Markov Decision Processes

- Policy: $S \rightarrow A$
- Every state is mapped to an action
- How should we evaluate a policy?
  - Evaluate the executions it may produce
- Need a function that determines the utility of an execution history
- Candidates:
  - Sum of utilities of states
  - Max state-utility in the execution
  - Min state-utility in the execution



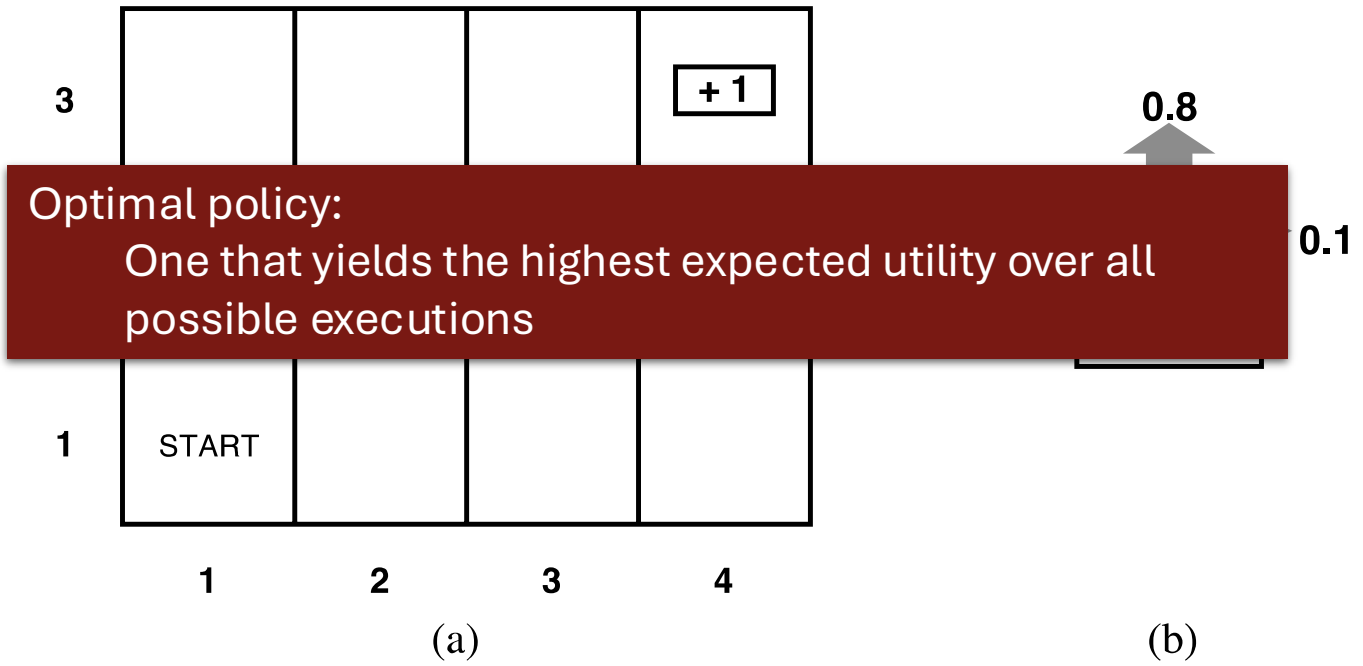
(a)



(b)

Markov Decision Processes

- Policy: $S \rightarrow A$
- Every state is mapped to an action
- How should we evaluate a policy?
  - Evaluate the executions it may produce
- Need a function that determines the utility of an execution history
- Candidates:
  - Sum of utilities of states
  - Max state-utility in the execution
  - Min state-utility in the execution



Optimal policy:
One that yields the highest expected utility over all possible executions

Stationary Utility Functions

- Stationary preferences lead to only two possible ways to assign utilities to sequences

  - Additive rewards
    - $U([s_0, \ldots, s_n]) = R(s_0) + \cdots + R(s_n)$
    - Problem: infinite horizon… infinite reward
    - How would we compare solutions?

  - Discounted rewards ($0 \leq \gamma \leq 1$)
    - $U([s_0, \ldots, s_n]) = R(s_0) + \gamma R(s_1) \ldots + \gamma^n R(s_n)$
    - Utility of an infinite sequence (bounded reward) is guaranteed to converge!

Solving MDPs

Solving MDPs

- Expected utility of using a policy π starting in s
  - Let $S_1, \ldots S_k, \ldots$ be the RVs denoting states generated at that timestep
  - $U^{\pi}(s) = E[\sum_t \gamma^t R(S_t)]$

- We want the "best" policy
  - $\pi^*(s) = argmax_{\pi} U^{\pi}(s)$

  Give me the policy that does the best, starting from s

- Should the optimal action depend on the timestep?

Solving MDPs

- Expected utility of using a policy π starting in s
  - Let $S_1, \ldots S_k, \ldots$ be the RVs denoting states generated at that timestep
  - $U^{\pi}(s) = E[\sum_t \gamma^t R(S_t)]$

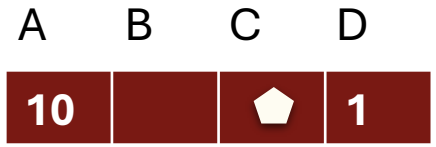
- We want the "best" policy
  - $\pi^*(s) = argmax_{\pi} U^{\pi}(s)$

  Give me the policy that does the best, starting from s

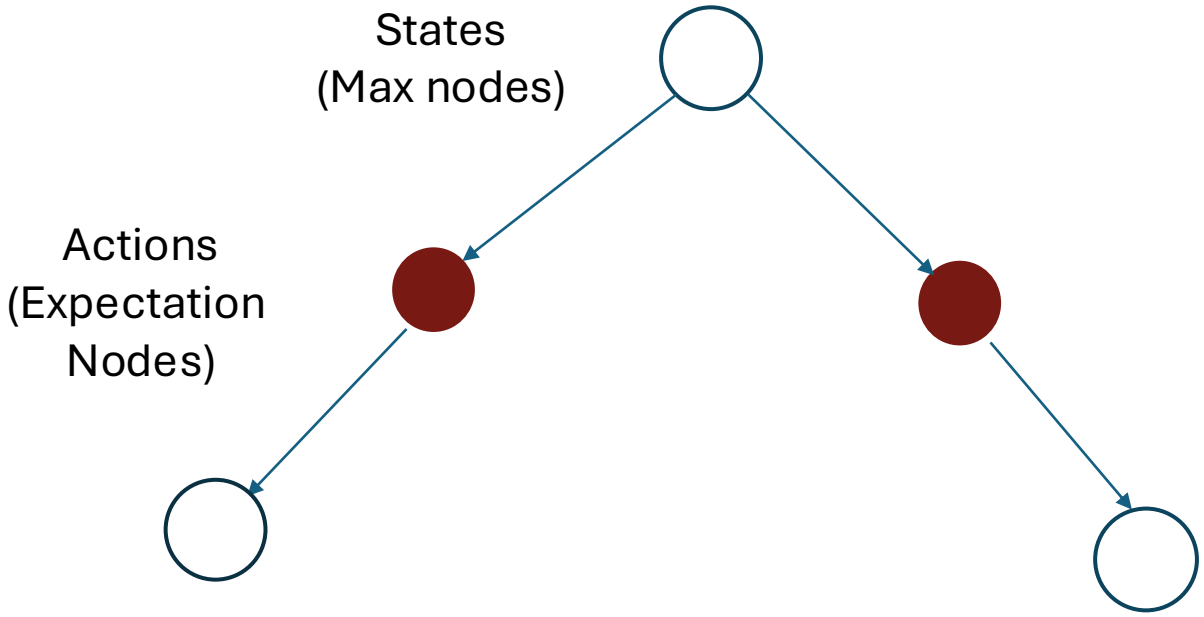
- For infinite horizon problems, π^* is independent of the starting state
  - i.e., the optimal action at a state doesn't depend on what you did before reaching it!

- $V(s)$: utility of a state under the optimal policy

MDP Search Tree: Discounting

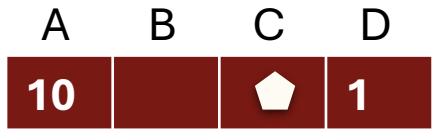


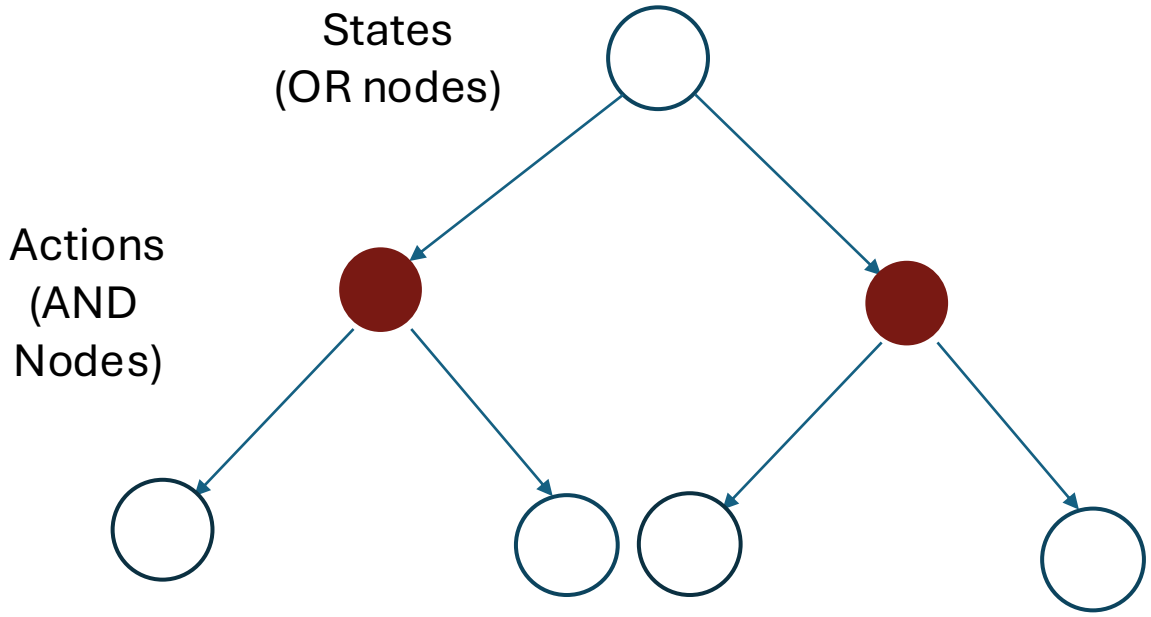
A, D: exit states
Transition probability: 0.8 move according to action executed, 0.2 stay



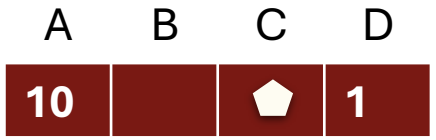
MDP Search Tree

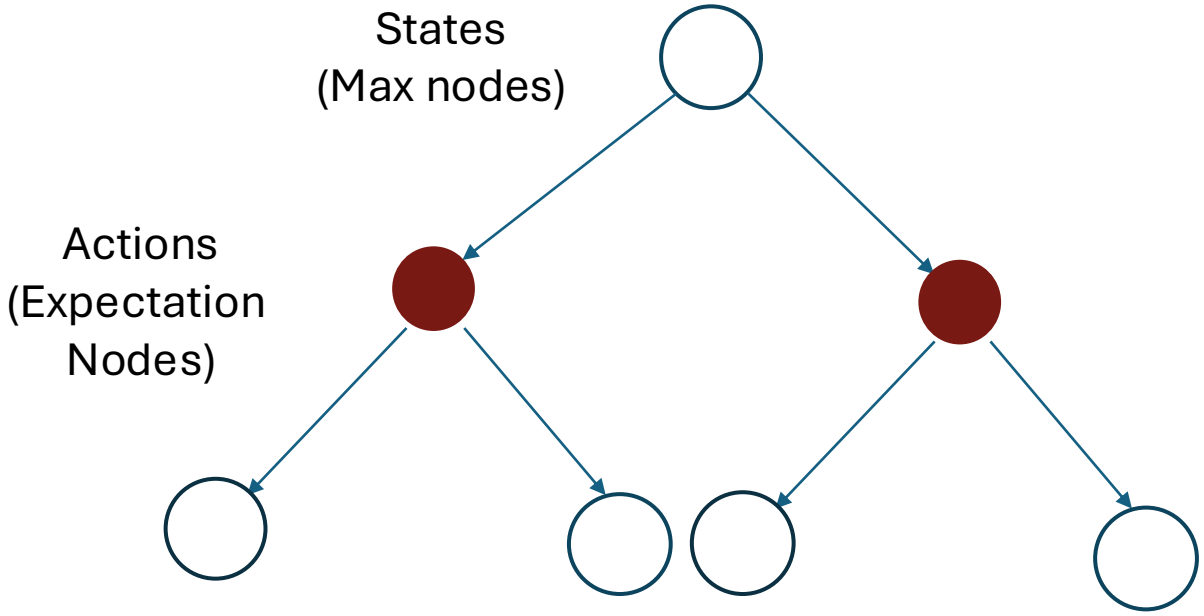
- Extend AND-OR trees with utility-theoretic principles





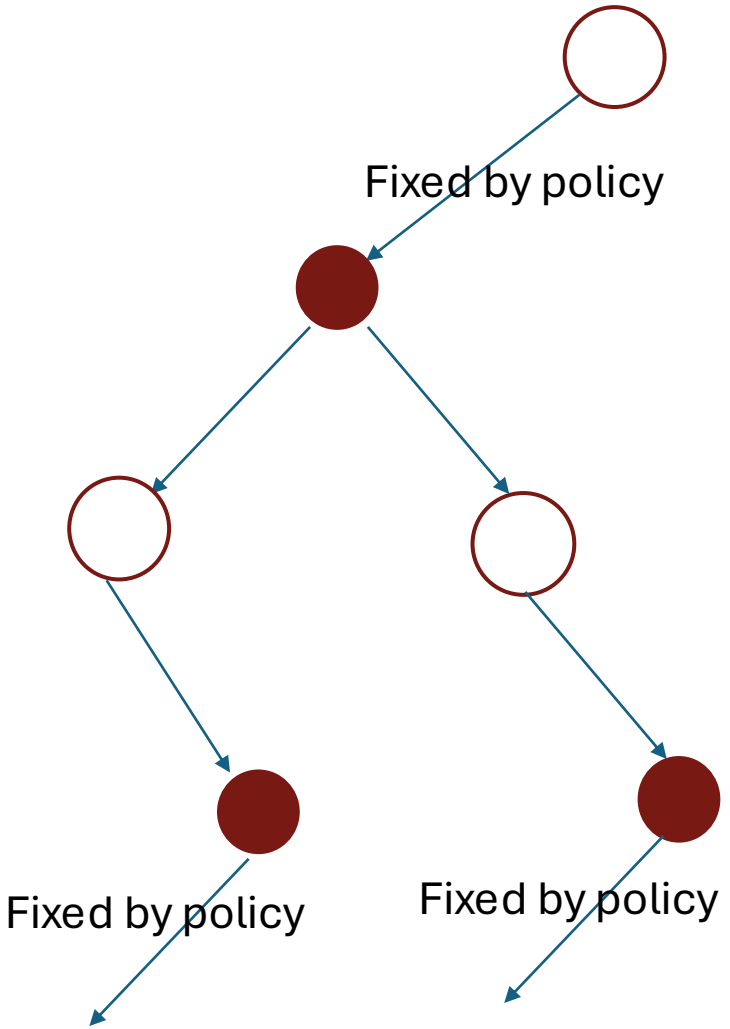
MDP Search Tree: Stochastic Transitions





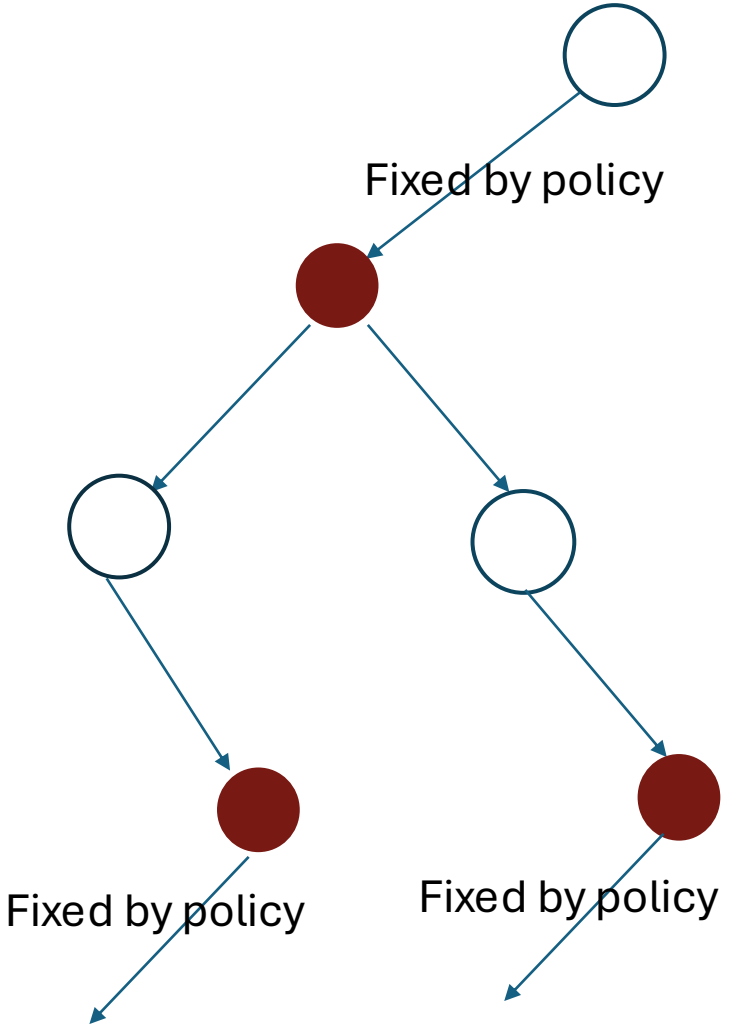
Optimal Policies

- 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 starting in s, executing a, then following π
- Optimal policy: π^*
  - $\pi^*(s) = \text{argmax}_{\pi} V^{\pi}(s)$
- Convenient:
  - Infinite horizon + discounting $\rightarrow \pi^*$ independent of starting state!



Optimal Policies

- Optimal policy: π^*
  - $\pi^*(s) = \text{argmax}_\pi V^\pi(s)$
- Optimal value of a state s
  - $V^{\pi^*}(s) = V(s) =$ expected utility starting in s and following π^*
- Q-function: value of (s, a) pair
  - $Q^{\pi^*}(s, a) = Q(s, a)$ expected utility starting in s, executing a, then following π^*



Computing Optimal Policies

- By definition of optimal policy:

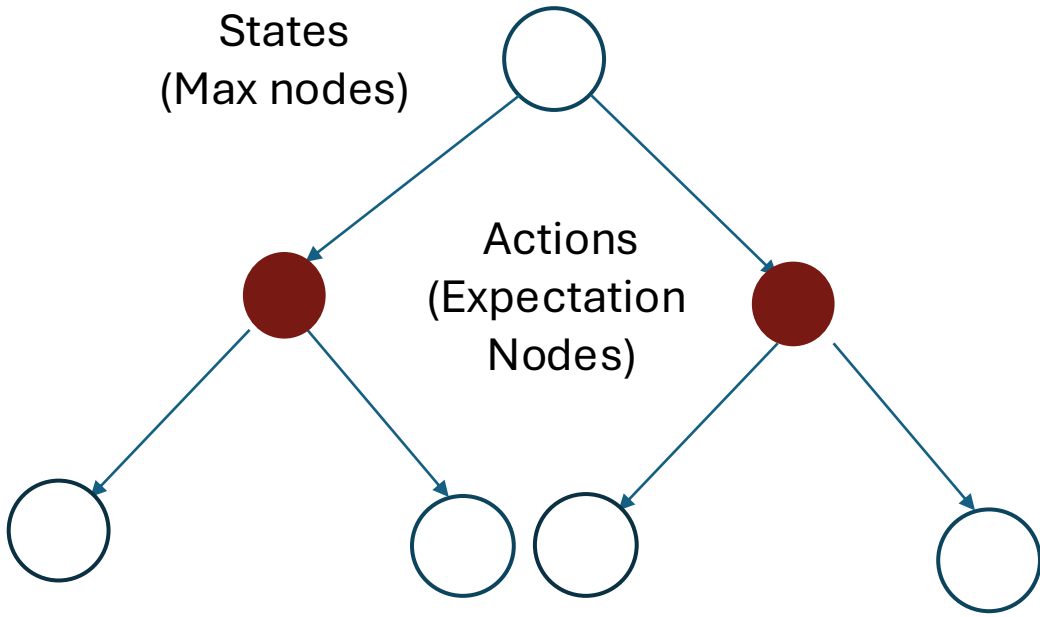
$$V^*(s) = \max_a Q^*(s,a)$$

- By definition of optimal Q function

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s) + \gamma V^*(s')]$$

- Thus

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

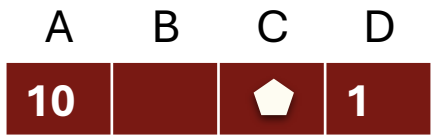


Bellman's Equation

How Would We Compute V^*?

- Problems:
  - Bellman equation is recursive
  - Tree is of unbounded depth, repetitive
  - All occurrences of B require recomputation of the same function

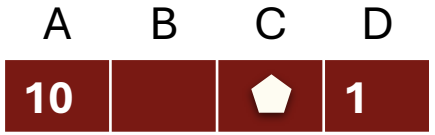
$$V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s) + \gamma V^*(s')\right]$$



- Tricks:
  - Incremental computation
  - Dynamic programming

A, D: exit states

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



1-step greedy policy over V_1:
Go left on B, right on C

V_2

V_1

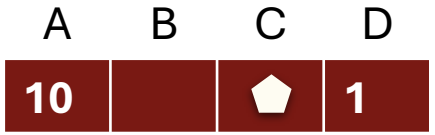
A	B	C	D
10	0	0	1

V_0

A	B	C	D
0	0	0	0

A, D: exit states

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



$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma V_k(s')]$$

$$V_2(s) = max\{ \sum_{s'} T(s, \leftarrow, s')[R(s) + \gamma V_1(s')],$$

$$\sum_{s'} T(s, \rightarrow, s')[R(s) + \gamma V_1(s')]\}$$

V_2

V_1

A	B	C	D
10	0	0	1

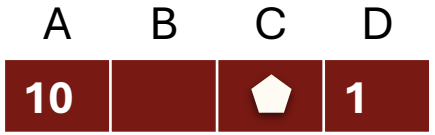
V_0

A	B	C	D
0	0	0	0

$$V_1(s) = \max_a \sum_{s'} T(s, a, s')[R(s)]$$

A, D: exit states

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



$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s) + \gamma V_k(s')]$$

$$V_2(s) = max\{ \sum_{s'} T(s,\leftarrow,s')[R(s) + \gamma V_1(s')],$$

$$\sum_{s'} T(s,\rightarrow,s')[R(s) + \gamma V_1(s')]\}$$

V_2

A	B	C	D
10	7.2	.72	1

V_1

A	B	C	D
10	0	0	1

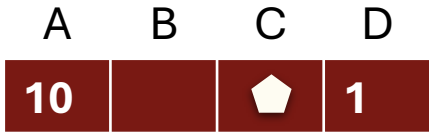
V_0

A	B	C	D
0	0	0	0

$$V_1(s) = \max_a \sum_{s'} T(s,a,s')[R(s)]$$

A, D: exit states

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



$$V_{k+1}(s) = max\{ \sum_{s'} T(s, \leftarrow, s')[R(s) + \gamma V_k(s')],$$

$$\sum_{s'} T(s, \rightarrow, s')[R(s) + \gamma V_k(s')]\}$$

V_3

A	B	C	D
10	8.5	5.3	1

V_2

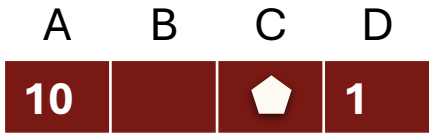
A	B	C	D
10	7.2	.72	1

V_1

A	B	C	D
10	0	0	1

V_0

A	B	C	D
0			0



$$V_{k+1}(s) = max\{ \sum_{s'} T(s, \leftarrow, s')[R(s) + \gamma V_k(s')],$$

$$\sum_{s'} T(s, \rightarrow, s')[R(s) + \gamma V_k(s')]\}$$

V_3

A	B	C	D
10	8.5	5.3	1

V_2

A	B	C	D
10	7.2	.72	1

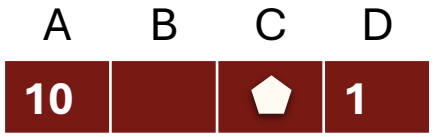
V_1

A	B	C	D
10	0	0	1

V_0

A	B	C	D
0			0

What happened here?



$$V_{k+1}(s) = max\{ \sum_{s'} T(s, \leftarrow, s')[R(s) + \gamma V_k(s')],$$

$$\sum_{s'} T(s, \rightarrow, s')[R(s) + \gamma V_k(s')]\}$$

V_4

A	B	C	D
10	8.7	7.1	1

V_3

A	B	C	D
10	8.5	5.3	1

V_2

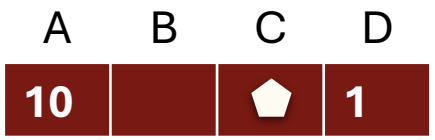
A	B	C	D
10	7.2	.72	1

V_1

A	B	C	D
10	0	0	1

V_0

A	B	C	D
0			0



$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s) + \gamma V_k(s')]$$

Time interpretation: k steps to go

V_4

A	B	C	D
10	8.7	7.1	1

V_3

A	B	C	D
10	8.5	5.3	1

V_2

A	B	C	D
10	7.2	.72	1

V_1

A	B	C	D
10	0	0	1

V_0

A	B	C	D
0			0

We've been implicitly iterating over the Q function as well:

$$Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s) + \gamma V_k(s')]$$

What is V^* anyway?

$$V^*(s_1) = \max_a \sum_{s_2} T(s_1, a, s_2)[R(s) + \gamma V^*(s_2)]$$

What we want: expected utility under best policy

$$V^*(s_1) = R(s_1) + \gamma \cdot \max_a \textstyle\sum_{s_2} T(s_1, a, s_2) \cdot \{$$

$$R(s_2) + \gamma \cdot \max_a \textstyle\sum_{s_3} T(s_2, a, s_3) \cdot \{$$

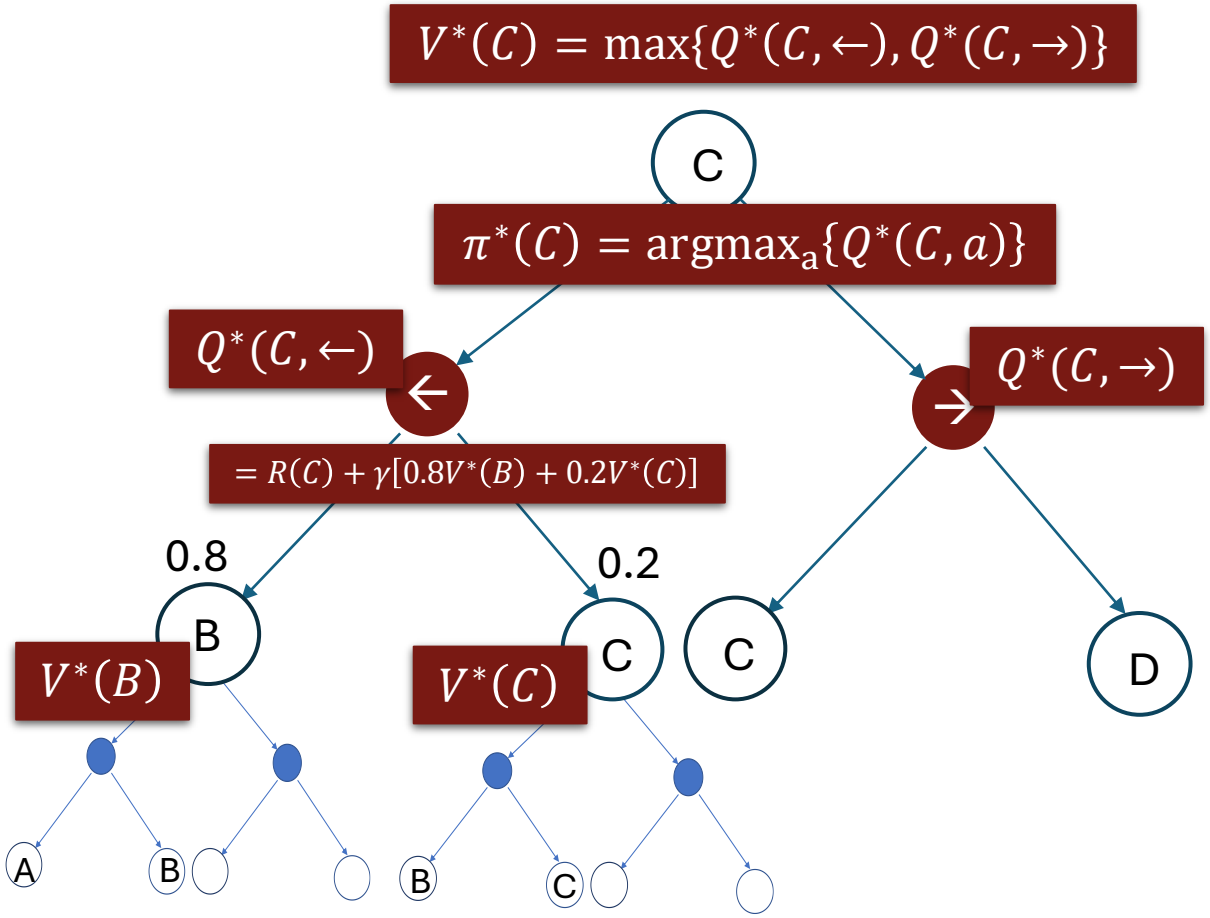
$$R(s_3) + \gamma \cdot \max_a \textstyle\sum_{s_4} T(s_3, a, s_4) \cdot \{ \ldots \}$$

Value obtained by acting optimally from s_4

$$R(s_3) + \gamma \cdot \max_a \textstyle\sum_{s_4} T(s_3, a, s_4) \cdot V^*(s_4)$$

$$R(s_2) + \gamma \cdot \max_a \textstyle\sum_{s_3} T(s_2, a, s_3)\, V^*(s_3)$$

$$R(s_1) + \gamma \cdot \max_a \textstyle\sum_{s_2} T(s_1, a, s_2) \cdot V^*(s_2)$$


$V^*(C) = \max\{Q^*(C, \leftarrow), Q^*(C, \rightarrow)\}$
C
$\pi^*(C) = \text{argmax}_a\{Q^*(C, a)\}$
$Q^*(C, \leftarrow)$
$Q^*(C, \rightarrow)$
$= R(C) + \gamma[0.8V^*(B) + 0.2V^*(C)]$
0.8
0.2
B
C
C
D
$V^*(B)$
$V^*(C)$
A
B
B
C

Value Iteration

- Break up the value computation into "time steps"
- $V_k(s)$ denotes the best value possible with k steps to go
- Start with $V_0(s) = 0$ no time steps left
- $V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma V_k(s')]$

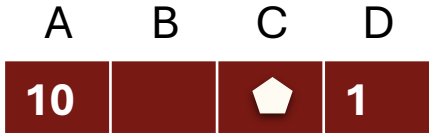
- Repeat until convergence (memorize $V_k(s)$ when it is first encountered)

- Theorem: Value iteration converges to the unique solution when $\gamma < 1$
- Why?

Computing Actions from Values

- Suppose we have the optimal values
- How should we act?
- We could use V* to compute best action:
  - $\pi^*(s) = argmax_a \sum_{s'} T(s,a,s') \left[R(s) + \gamma V^*(s')\right]$
- This is called policy extraction
- More efficient: store Q*

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



V_4

Ⓐ	Ⓑ	Ⓒ	Ⓓ
10	8.7	7.1	1

Limitations of Value Iteration

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma V_k(s')]$$

- $O(S^2 A)$ 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

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: go greedy over one-step lookahead converged (but not optimal!) utilities as future values`
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions