AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Markov decision Process -Value Iteration

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

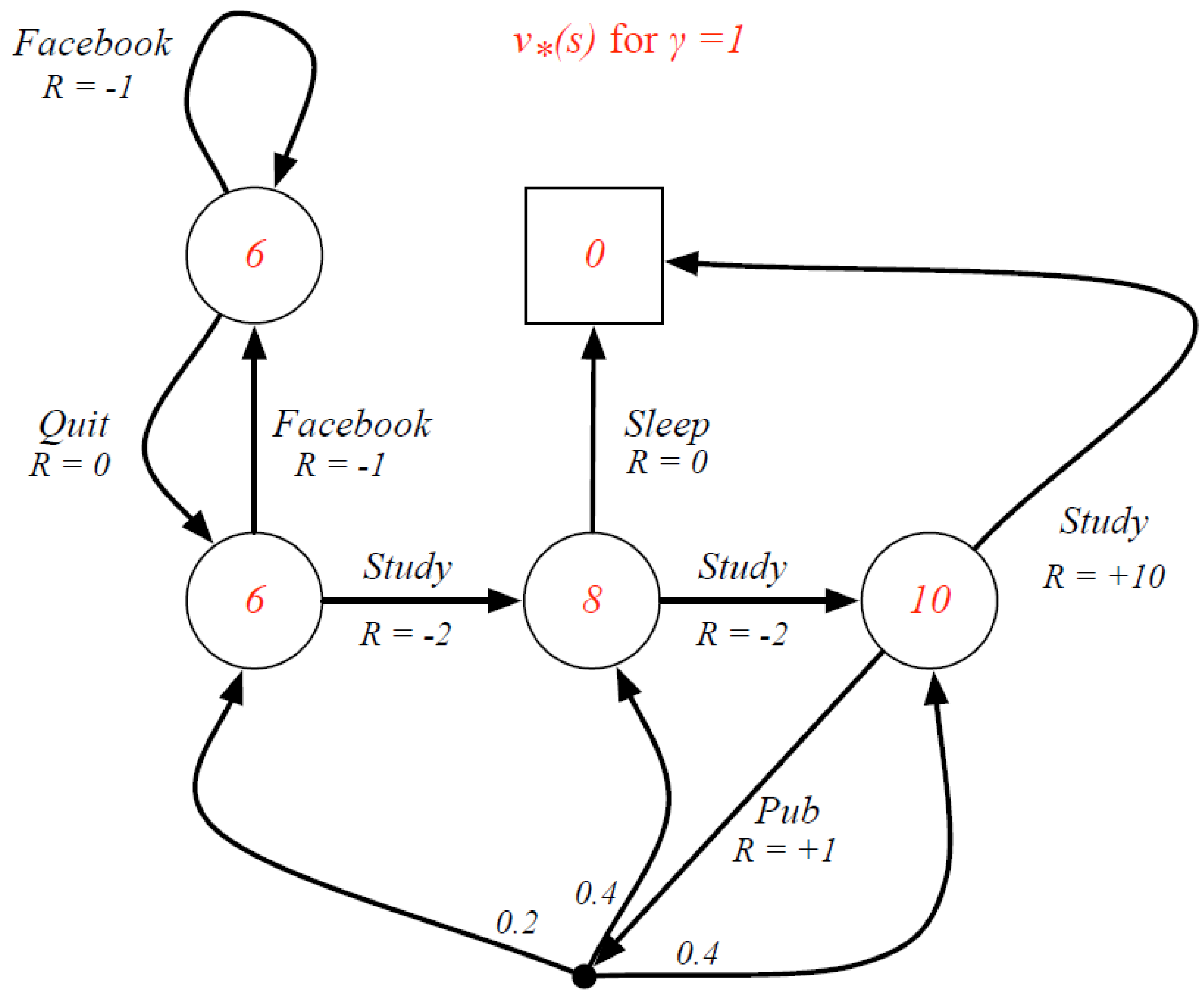
$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

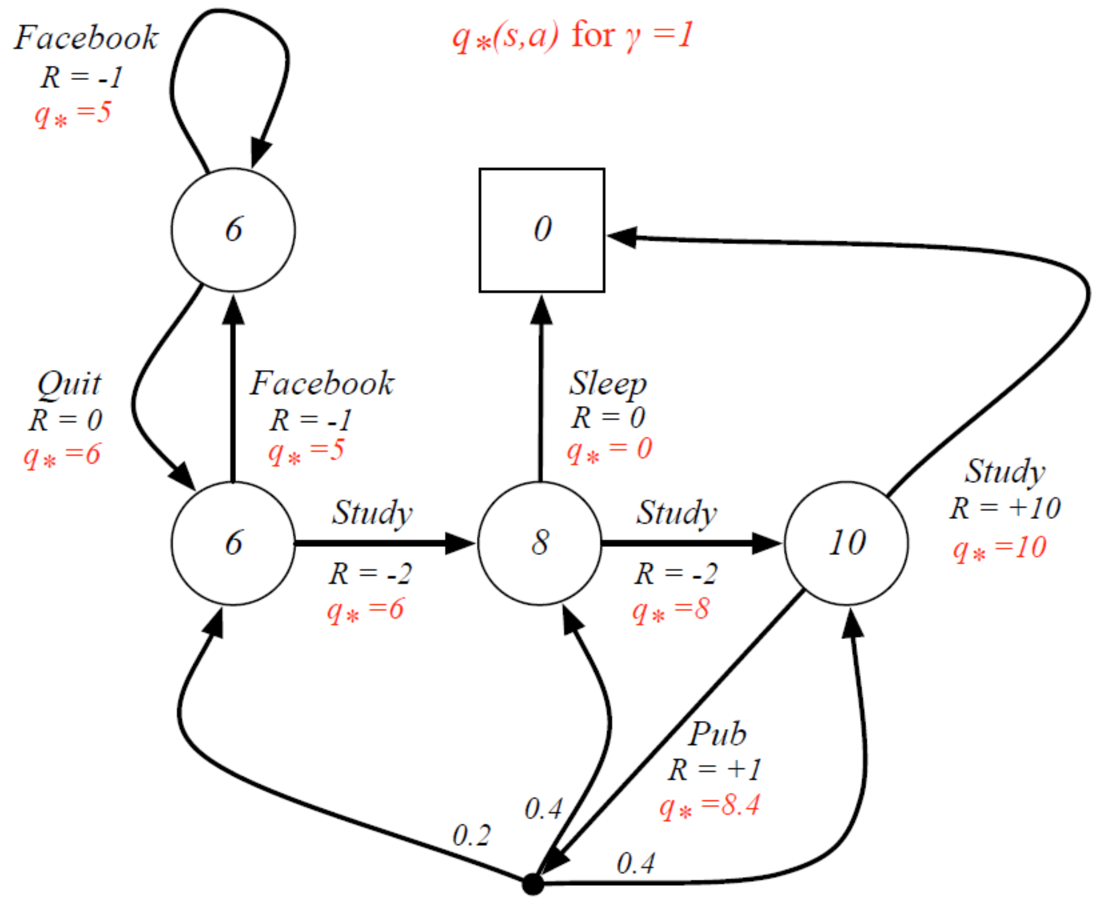
$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Example: Optimal Value Function for Student MDP



Example: Optimal Action-Value Function for Student MDP



Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

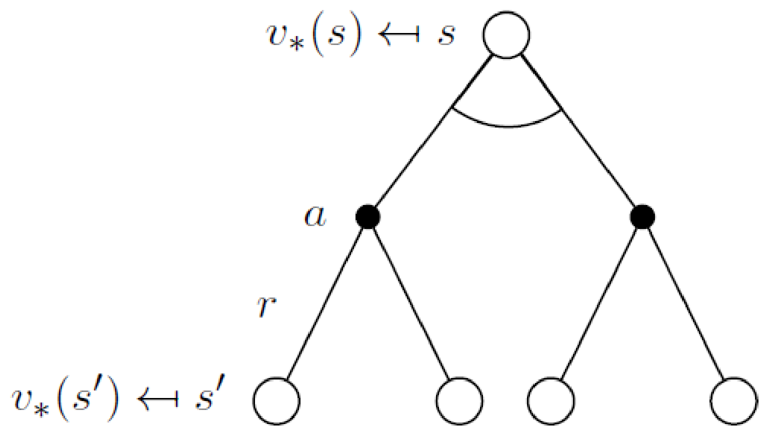
Finding an Optimal Policy

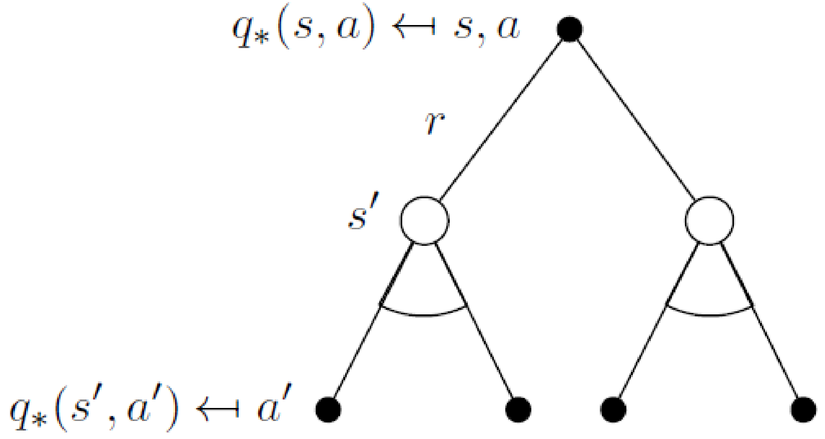
An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s, a) \\ 0 & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Optimality Equations for V*(2) and Q*(2)





$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$$

value function for optimal policy can be solved using dynamic programming

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
    - Value Iteration
    - Policy Iteration
    - Q-learning
    - Sarsa

What is Dynamic Programming (DP)?

- Algorithm design technique for optimization problems
- Divide the problem into a set of smaller subproblems
- Solves problems by combining solutions of subproblems.
- Overlapping subproblems
- Can be applied to problems that requires the recomputation of values to get final solution.

Dynamic Programming Approaches

- **Top down approach**
  - Divide into small sub-problems
  - If subproblem already solved, use that solution
  - Otherwise, the solution is memoized: to speed upcomputation by storing the results and returning cached result when the same the inputs occur again.
- **Bottom up approach**
- solutions to small problems are calculated which combined to generate the solutionof the problem.

Steps in Dynamic Programming

1. Characterize structure of an optimal solution.
2. Define value of optimal solution recursively.
3. Compute optimal solution values either top-down or bottom-up approach
4. Construct an optimal solution from computed values.

Dynamic Programming in RL

- Dynamic programming: combine solutions to the sub-problems
    - Optimal substructure
    - Overlapping sub-problems

Goal of RL is to find a policy that maximize the expected reward

Markov Decision Processes

How RL satisfies DP requirements?

- Markov decision processes satisfy:
- Optimal substructure: recursive decomposition
- Overlapping sub-problems: Value function stores and reuses solutions

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$

Planning by Dynamic Programming

- Dynamic programming requires full knowledge of the MDP
- It is used for planning in an MDP

•**Prediction (Policy Evaluation):**

- **Input:** An MDP + a fixed policy π.
- **Output:** The value function $v_\pi(s)$,i.e., the expected return under that policy.
- Question answered: *"If I follow this policy, how good will it be?"*

•**Control (Policy Improvement):**

- **Input:** Just the MDP (no fixed policy given).
- **Output:** The optimal value function $v^*(s)$and the optimal policy π^*.
- Question answered: *"What is the best policy I can follow?"*

if you start optimally, and then always continue optimally, the overall behavior is optimal.

How to find optimal policy?

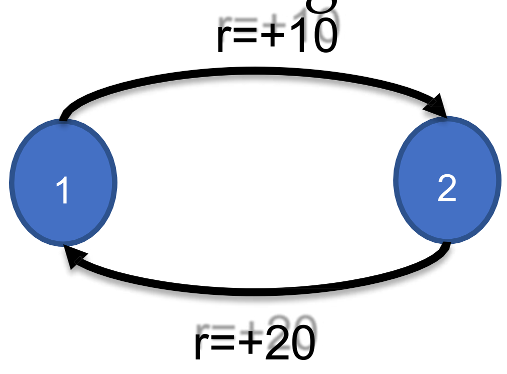
- Problem: Find the optimal policy π^* for a given MDP.
  - **Solution: Iterative application of Bellman optimality equation**

- Principle of Optimality
  - **Any optimal policy can be subdivided into two components which make the overall behaviour optimal**
    - An optimal first action $a*$ followed by an optimal policy from successor state S
    - A policy $\pi(a|s)$ achieves the optimal value from state s, $v\_\pi(s) = v*(s)$, if and only if:
      - state s' reachable from s
      - π achieves the optimal value from state s', $v\_\pi(s')=v*(s')$

Value Iteration Algorithm

Value Iteration Algorithm

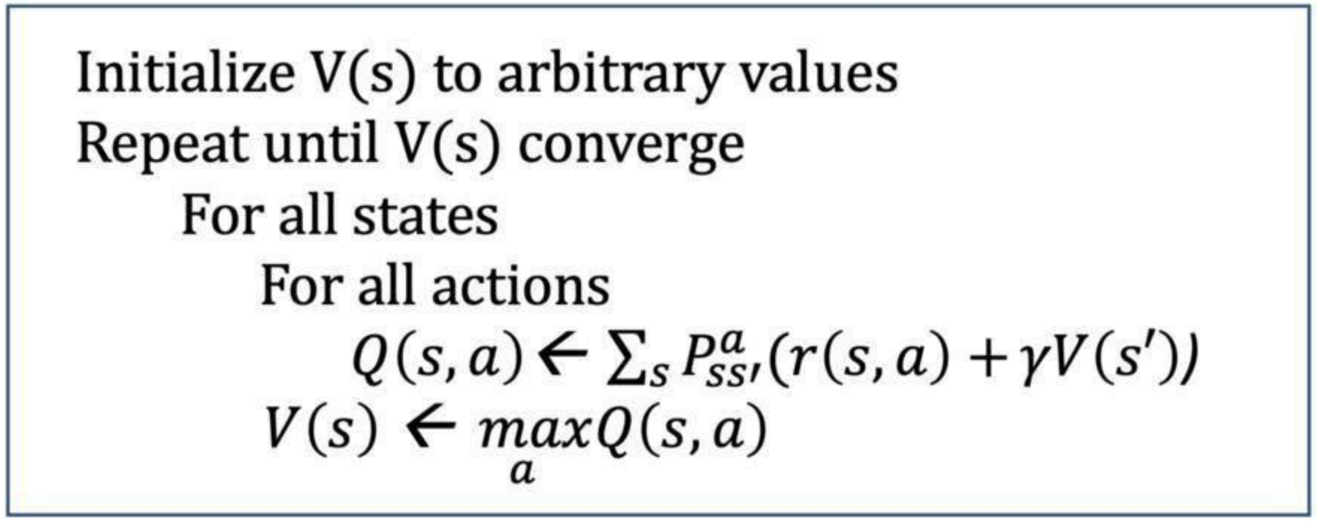
- To calculate the values of the states of MDP, with known transition probabilities and rewards.
- Value Iteration algorithm computes the optimal state value function by iteratively improving the estimate of *V*(s).
- The algorithm initializes *V*(s) to arbitrary random values. It repeatedly updates the *Q(s, a)* and *V*(s) values until they converge.
- Value Iteration is guaranteed to converge to the optimal values.



$$V(1) = 10 + \gamma\left(20 + \gamma\left(10 + \gamma\left(20 + \dots\right)\right)\right) = \sum_{i=0}^{\infty} 10\gamma^{2i} + 20\gamma^{2i+1}$$

$$V_*(s) = \max_a \sum_{s'} p_{ss'}^a \left(r(s,a) + \gamma v_*(s')\right)$$

Value Iteration Algorithm

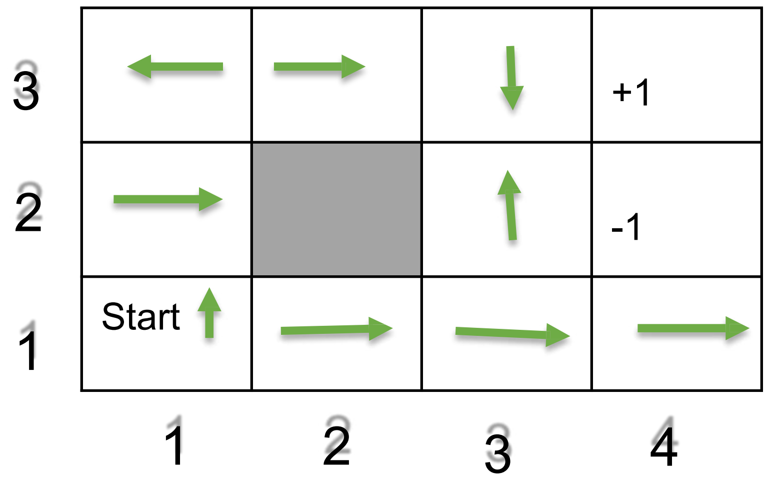


```
Initialize V(s) to arbitrary values
Repeat until V(s) converge
    For all states
        For all actions
            $Q(s,a) \leftarrow \sum_s P^a_{ss'}(r(s,a) + \gamma V(s'))$
        $V(s) \leftarrow \max_a Q(s,a)$
```

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s' | s, a) V(s')$$

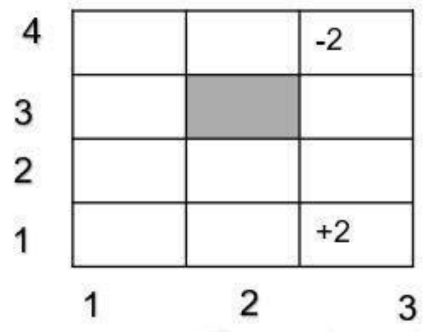
MDPs: Policy

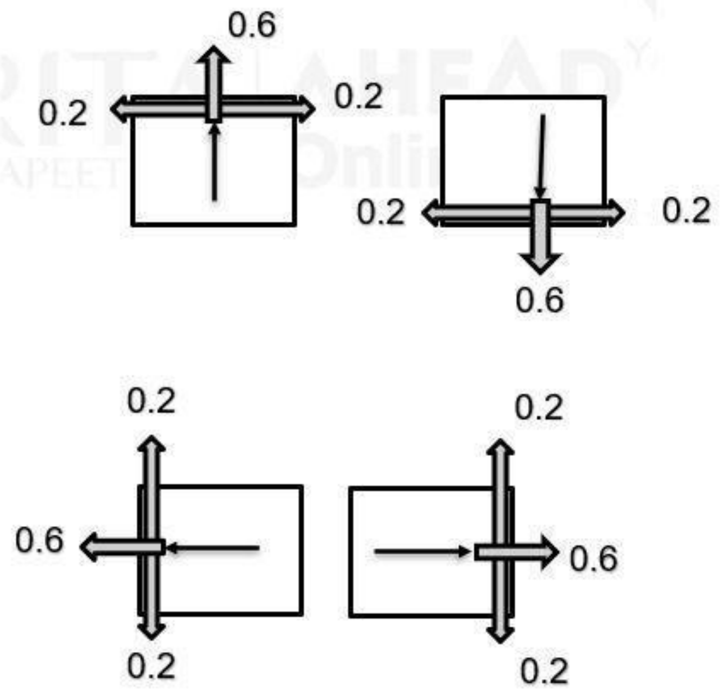
- The robot needs to know what to do as the decision process unfolds…
- It starts in a state, selects an action, ends up in another state selects another action….
- Needs to make the same decision over and over: Given the current state what should I do?
- So a policy for an MDP is a single decision function *π(s)* that specifies what the agent should do for each state *s*



Example MDP

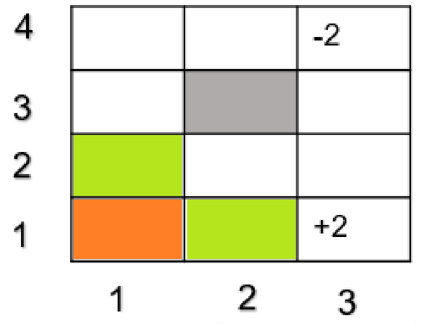
- Environment 3 x 4 grid, one blocked state – 11 states
- Two terminal states: (3, 4), (3, 1)
- Actions: up, down, left, right
- 0.6 to reach extended effect
- 0.2 probability to move at right angle of extended direction
- If the agents bumps into a wall, it stays there
- Reward:
  - For terminal states +/- 2
  - Other states: -0.5



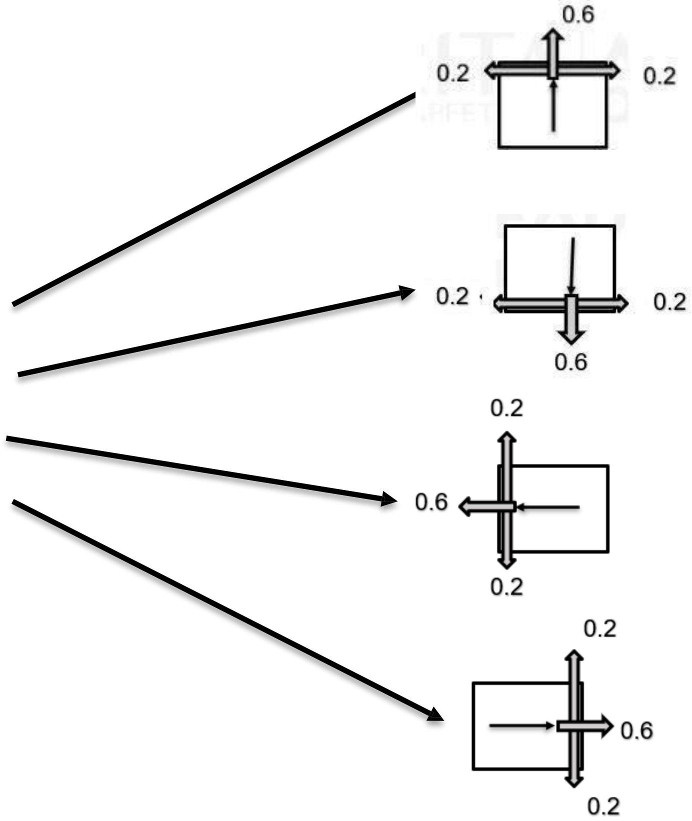


Value Computation: state (1, 1)

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} P(s' \mid s, a) V(s')$$



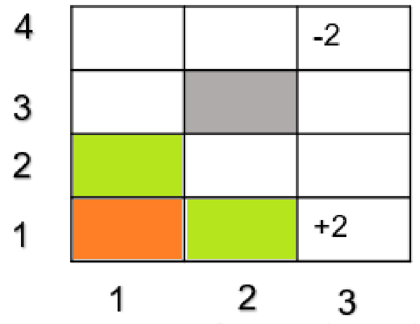
$$V(1,1) = -0.5 + \max \left\{ \begin{array}{l} 0.6 \times v(1,2) + 0.2 \times v(1,1) + 0.2 \times v(2,1) \\ \\ \\ \\ \end{array} \right. \quad \begin{array}{l} \text{UP} \\ \text{DOWN} \\ \text{LEFT} \\ \text{RIGHT} \end{array}$$



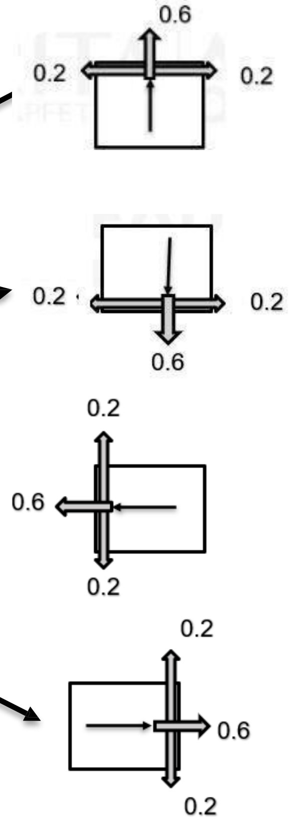
$$= -0.5 + \max \left\{ \begin{array}{l} 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ \\ \\ \\ \end{array} \right. \qquad = -0.5 + \max \left\{ \begin{array}{l} 0 \\ \\ \\ \\ \end{array} \right.$$

Value Computation: state (1, 1)

$$V(s_1) = R(s_1) + \gamma \max_a \sum_{s'} P(s' \mid s_1, a) V(s')$$



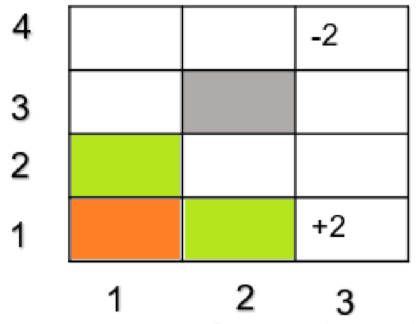
$$V(1,1) = -0.5 + \max{}^{*}1 \begin{cases} 0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{UP} \\ 0.6 \times V(1,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{DOWN} \\ & \text{LEFT} \\ & \text{RIGHT} \end{cases}$$



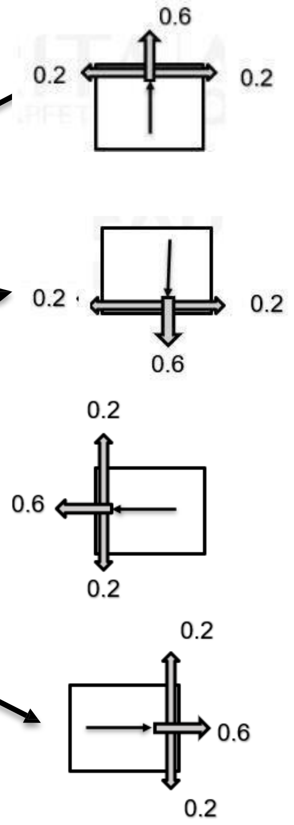
$$= -0.5 + \max \begin{cases} 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ \\ \end{cases} \qquad = -0.5 + \max \begin{cases} 0 \\ 0 \\ \\ \end{cases}$$

Value Computation: state (1, 1)

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s')$$



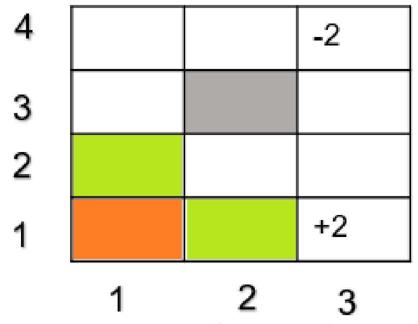
$$V(1,1) = -0.5 + \max{}^*1 \begin{cases} 0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{UP} \\ 0.6 \times V(1,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{DOWN} \\ 0.6 \times V(1,1) + 0.2 \times V(1,2) + 0.2 \times V(1,1) & \text{LEFT} \\ & \text{RIGHT} \end{cases}$$



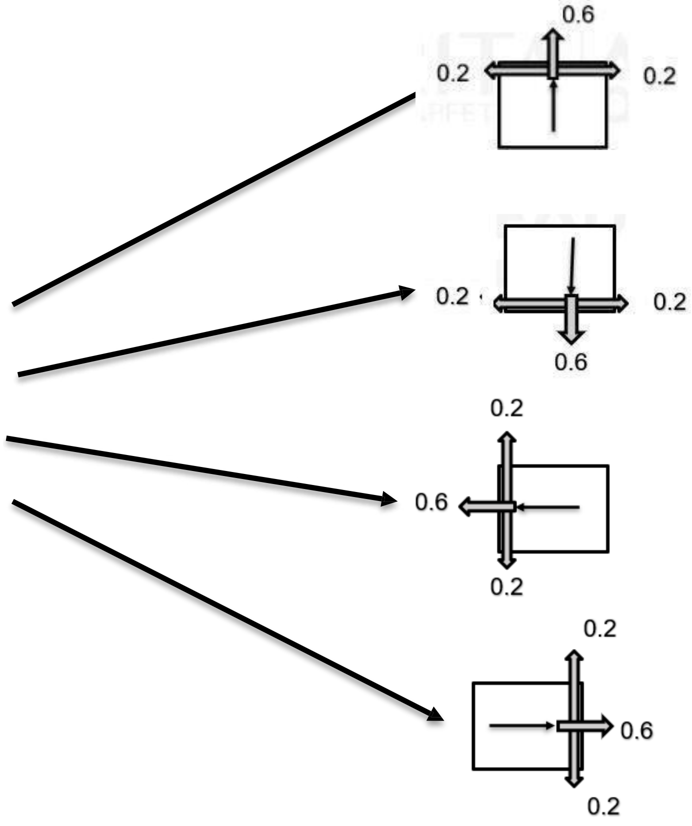
$$= -0.5 + \max \begin{cases} 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \end{cases} \quad = -0.5 + \max \begin{cases} 0 \\ 0 \\ 0 \end{cases}$$

Value Computation: state (1, 1)

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s')$$



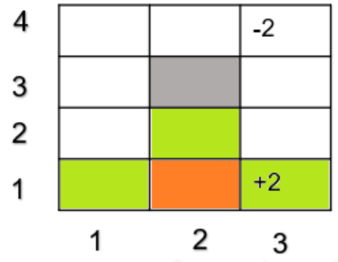
$$V(1,1) = -0.5 + \max{}^*1 \begin{cases} 0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{UP} \\ 0.6 \times V(1,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{DOWN} \\ 0.6 \times V(1,1) + 0.2 \times V(1,2) + 0.2 \times V(1,1) & \text{LEFT} \\ 0.6 \times V(2,1) + 0.2 \times V(1,2) + 0.2 \times V(1,1) & \text{RIGHT} \end{cases}$$



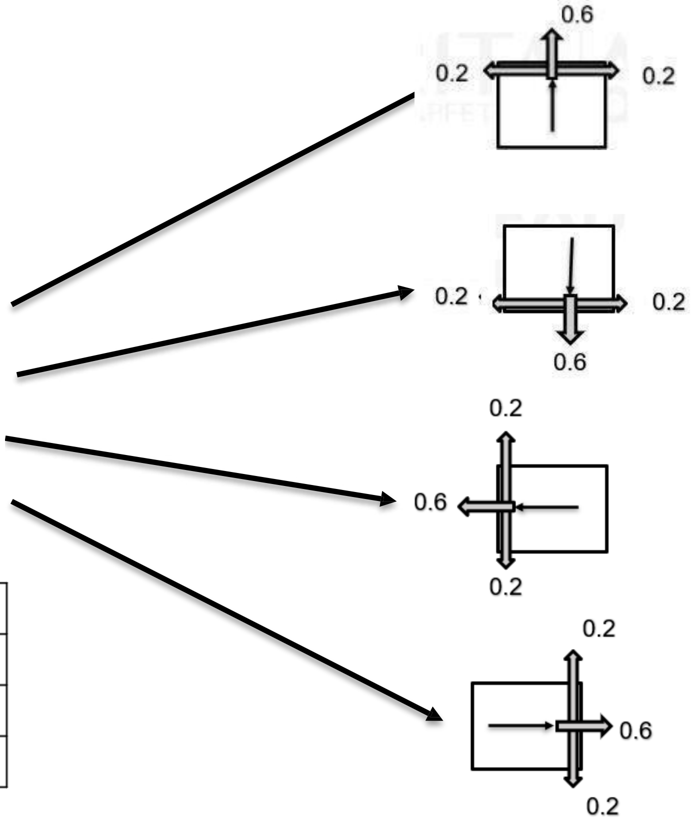
$$= -0.5 + \max \begin{cases} 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \end{cases} = -0.5 + \max \begin{cases} 0 \\ 0 \\ 0 \\ 0 \end{cases} = -0.5$$

Value Computation: state (2, 1)

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V(s')$$



$$V(2,1) = -0.5 + \max \begin{cases} 0.6 \times v(2,2) + 0.2 \times v(1,1) + 0.2 \times v(3,1) & \text{UP} \\ 0.6 \times V(2,1) + 0.2 \times v(1,1) + 0.2 \times v(3,1) & \text{DOWN} \\ 0.6 \times v(1,1) + 0.2 \times v(2,2) + 0.2 \times v(2,1) & \text{LEFT} \\ 0.6 \times v(3,1) + 0.2 \times v(2,2) + 0.2 \times v(2,1) & \text{RIGHT} \end{cases}$$



$$= -0.5 + \max \begin{cases} 0.4 \\ 0.4 \\ 0.0 \\ 1.2 \end{cases} = -0.5 + 1.2 = 0.7$$

	1	2	3
4			
3			
2			
1	-0.5	0.7	

V(3, 2)

4			
3			
2	0.7		0.7
1	-0.5		
	1	2	3

$$V(3,\ 2) = -0.5 + \max \begin{cases} UP & 0.6 \times V(3,3) + 0.2 \times V(2,2) + 0.2 \times V(3,2) \\ DOWN & 0.6 \times V(3,1) + 0.2 \times V(2,2) + 0.2 \times V(3,2) \\ LEFT & 0.2 \times V(3,3) + 0.6 \times V(2,\ 2) + 0.2 \times V(3,1) \\ RIGHT & 0.6 \times V(3,2) + 0.2 \times V(3,3) + 0.2 \times V(3,1) \end{cases}$$

$$= -0.5 + \max \begin{cases} \text{UP} & 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ \text{DOWN} & 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \\ \text{LEFT} & 0.2 \times 0 + 0.6 \times 0 + 0.2 \times 0 \\ \text{RIGHT} & 0.6 \times 0 + 0.2 \times 0 + 0.2 \times 0 \end{cases} = -0.5 + \max \begin{cases} 0 \\ 1.2 \\ 0.4 \\ 0.4 \end{cases} = 0.7$$

V(1, 4), V(2, 2)

$$V(1,4) = -0.5 + 1 * \max \begin{cases} \text{UP} & 0.6 \times V(1,4) + 0.2 \times V(1,4) + 0.2 \times V(2,4) \\ \text{DOWN} & 0.6 \times V(1,3) + 0.2 \times V(2,4) + 0.2 \times V(1,4) \\ \text{LEFT} & 0.6 \times V(2,4) + 0.2 \times V(1,4) + 0.2 \times V(1,3) \\ \text{RIGHT} & 0.6 \times V(1,4) + 0.2 \times V(1,4) + 0.2 \times V(1,3) \end{cases}$$

= -0.5

4	-0.5		
3			
2	0.7	-0.5	0.7
1	-0.5		
	1	2	3

$$V(2,2) = -0.5 + 1 * \max \begin{cases} \text{UP} & 0.6 \times V(2,2) + 0.2 \times V(3,2) + 0.2 \times V(1,2) \\ \text{DOWN} & 0.6 \times V(2,1) + 0.2 \times V(1,2) + 0.2 \times V(3,2) \\ \text{LEFT} & 0.6 \times V(1,2) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \\ \text{RIGHT} & 0.6 \times V(3,2) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \end{cases}$$

$$= -0.5 + 1 * \max \begin{cases} 0 \\ 0 \\ 0 \\ 0 \end{cases} \quad = -0.5$$

V(2, 4), V(2, 2)

$$V(2,4) = -0.5 + 1 * \max \begin{cases} \text{UP} & 0.6 \times V(2,4) + 0.2 \times V(1,4) + 0.2 \times V(3,4) \\ \text{DOWN} & 0.6 \times V(2,4) + 0.2 \times V(1,4) + 0.2 \times V(3,2) \\ \text{LEFT} & 0.6 \times V(3,4) + 0.2 \times V(2,4) + 0.2 \times V(3,3) \\ \text{RIGHT} & 0.6 \times V(1,4) + 0.2 \times V(2,2) + 0.2 \times V(2,4) \end{cases}$$

$$= -0.8$$

4	-0.5	-0.8	
3			
2	0.7	-0.5	
1	-0.5		
	1	2	3

$$V(2,2) = -0.5 + 1 * \max \begin{cases} \text{UP} & 0.6 \times V(3,4) + 0.2 \times V(3,3) + 0.2 \times V(3,3) \\ \text{DOWN} & 0.6 \times V(3,2) + 0.2 \times V(3,3) + 0.2 \times V(3,2) \\ \text{LEFT} & 0.6 \times V(2,3) + 0.2 \times V(3,4) + 0.2 \times V(3,2) \\ \text{RIGHT} & 0.6 \times V(3,3) + 0.2 \times V(2,4) + 0.2 \times V(3,2) \end{cases}$$

$$= -0.5 + 1 * \max \begin{cases} 0 \\ 0 \\ 0 \\ 0 \end{cases} \qquad = -0.5$$

After First iteration

-0.5	-0.8	-2
-0.5	▨	-0.8
-0.5	-0.5	0.7
-0.5	0.7	+2

Second iteration V(1, 1)

-0.5	-0.8	-2
-0.5	▨	-0.8
-0.5	-0.5	0.7
-0.5	0.7	+2

$$V(1,1) = -0.5 + \max{}^{*}1 \begin{cases} \text{UP} & 0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1) \\ \text{DOWN} & 0.6 \times V(1,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) \\ \text{LEFT} & 0.6 \times V(1,1) + 0.2 \times V(1,2) + 0.2 \times V(2,1) \\ \text{RIGHT} & 0.6 \times V(2,2) + 0.2 \times V(1,2) + 0.2 \times V(1,1) \end{cases}$$

$$= -0.5 + \max \begin{cases} -0.26 \\ -0.66 \\ -0.50 \\ 0.22 \end{cases} = -0.28$$

Second iteration

$$V(2,1) = -0.5 + \max{}^{*}1 \begin{cases} \text{UP } 0.6 \times V(2,2) + 0.2 \times V(1,1) + 0.2 \times V(2,4) \\ \text{DOWN } 0.6 \times V(2,1) + 0.2 \times V(1,1) + 0.2 \times V(3,2) \\ \text{LEFT } 0.6 \times V(1,1) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \\ \text{RIGHT } 0.6 \times V(3,1) + 0.2 \times V(2,2) + 0.2 \times V(2,1) \end{cases} = -0.8$$

$$= -0.5 + \max{}^{*}1 \begin{cases} 0 \\ 0.72 \\ -0.26 \\ -0.08 \end{cases} = -0.5 + 0.72 = 0.22$$

$$V(1,2) = -0.5 + 1^{*}\max \begin{cases} \text{UP } 0.6 \times V(1,3) + 0.2 \times V(1,2) + 0.2 \times V(1,2) \\ \text{DOWN } 0.6 \times V(1,2) + 0.2 \times V(1,2) + 0.2 \times V(2,2) \\ \text{LEFT} \\ \text{RIGHT} \end{cases}$$

Second iteration

$$V(2,2) = -0.5 + \max{}^*1 \begin{cases} \text{UP } 0.6 \times V(2,2) + 0.2 \times V(1,2) + 0.2 \times V(3,2) \\ \text{DOWN } 0.6 \times V(2,1) + 0.2 \times V(1,2) + 0.2 \times V(2,2) \\ \text{LEFT } 0.6 \times V(1,2) + 0.2 \times V(2,2) + 0.2 \times V(2,2) \\ \text{RIGHT } 0.6 \times V(3,2) + 0.2 \times V(2,2) + 0.2 \times V(2,2) \end{cases} = -0.8$$

$$= -0.5 + \max \begin{cases} -0.3-0.10+0.14 \\ -0.42-0.10+0.10 \\ -0.30-0.10-0.14 \\ 0.42-0.10-0.14 \end{cases}$$

$$= -0.5 + \max \begin{cases} \text{UP } -0.26 \\ \text{DOWN } 0.22 \\ \text{LEFT } -0.26 \\ \text{RIGHT } 0.46 \end{cases} \qquad = -0.5 + 0.16 = -0.04$$

Second iteration

$$V(3,2) = -0.5 + \max{}^*1 \begin{cases} \text{UP } 0.6 \times V(3,3) + 0.2 \times V(3,2) + 0.2 \times V(3,2) \\ \text{DOWN } 0.6 \times V(3,1) + 0.2 \times V(2,2) + 0.2 \times V(3,2) \\ \text{LEFT } 0.6 \times V(2,2) + 0.2 \times V(3,3) + 0.2 \times V(3,1) \\ \text{RIGHT } 0.6 \times V(3,2) + 0.2 \times V(5,3) + 0.2 \times V(3,1) \end{cases}$$

$$= -0.5 + \max \begin{cases} -0.48-0.10+0.14 \\ -0.12-0.10+0.14 \\ -0.30-0.10-0.40 \\ 0.42-0.10-0.40 \end{cases} \qquad = -0.5 + 0.66 = -0.16$$

Second iteration

$$V(3,3) = -0.5 + \max \begin{cases} \text{UP } 0.6 \times V(3,4) + 0.2 \times V(3,3) + 0.2 \times V(3,3) \\ \text{DOWN } 0.6 \times V(3,2) + 0.2 \times V(3,3) + 0.2 \times V(3,3) \\ \text{LEFT } 0.6 \times V(3,3) + 0.2 \times V(3,4) + 0.2 \times V(3,2) \\ \text{RIGHT } 0.6 \times V(3,3) + 0.2 \times V(3,4) + 0.2 \times V(3,2) \end{cases}$$

$$= -0.5 + \max \begin{cases} 0.6\text{x}-2+0.2\text{x}-0.8+0.2\text{x}-0.8 \\ 0.6\text{x}0.7+0.2\text{x}-0.8+0.2\text{x}-0.8 \\ 0.6\text{x}-0.8+0.2\text{x}-2+0.2\text{x}0.7 \\ 0.6\text{x}-0.8+0.2\text{x}-2+0.2\text{x}0.7 \end{cases} = -0.5 + \max \begin{cases} -1.52 \\ 0.1 \\ 0.22 \\ 0.22 \end{cases}$$

$$= -0.5 + 0.22 = --0.21$$

Second iteration

$$V(2,4) = -0.5 + \max{}^{*}1 \begin{cases} \text{UP } 0.6 \times V(2,4) + 0.2 \times V(1,4) + 0.2 \times V(3,4) \\ \text{DOWN } 0.6 \times V(2,4) + 0.2 \times V(1,4) + 0.2 \times V(3,4) \\ \text{LEFT } 0.6 \times V(1,4) + 0.2 \times V(2,4) + 0.2 \times V(3,4) \\ \text{RIGHT } 0.6 \times V(3,4) + 0.2 \times V(2,4) + 0.2 \times V(2,4) \end{cases}$$

$$= -0.5 + \max \begin{cases} -0.98 \\ -0.98 \\ -0.6 \\ -0.49 \end{cases}$$

$$= -0.44$$

Second iteration

$$V(1,3) = -0.5 + \max{}^{*}1 \begin{cases} \text{UP } 0.6 \times V(1,4) + 0.2 \times V(1,3) + 0.2 \times V(1,3) \\ \text{DOWN } 0.6 \times V(1,2) + 0.2 \times V(1,3) + 0.2 \times V(1,3) \\ \text{LEFT } 0.6 \times V(1,3) + 0.2 \times V(1,4) + 0.2 \times V(1,2) \\ \text{RIGHT } 0.6 \times V(1,3) + 0.2 \times V(1,4) + 0.2 \times V(1,2) \end{cases}$$

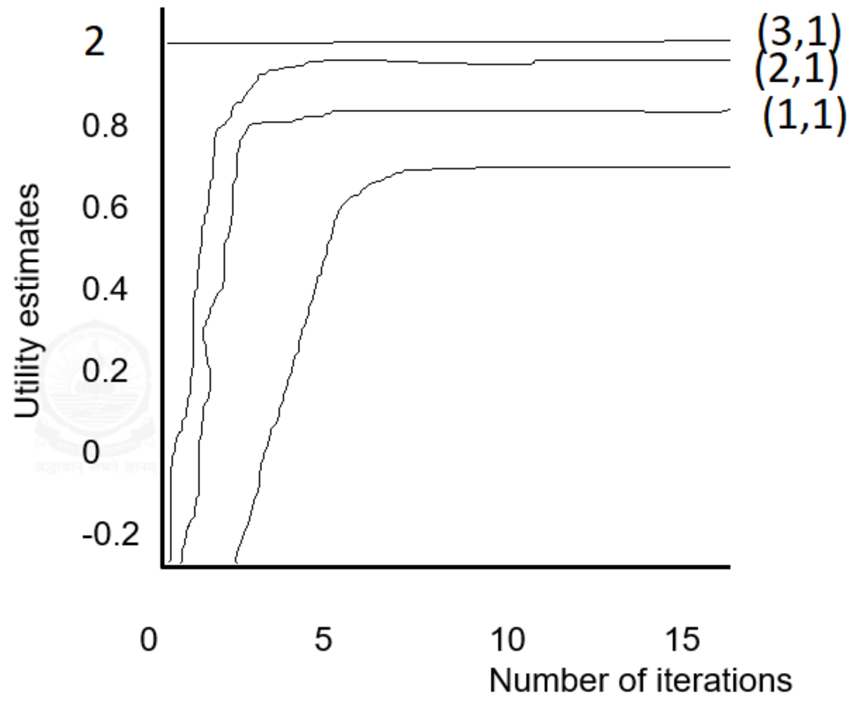
-1.0	-0.44	-2
-1.0	▨	-0.28
-1.00	-0.04	0.16
-0.35	0.22	+2

$$= -0.5 + \max{}^{*}1 \begin{cases} \text{UP } -0.30-0.10-0.10 \\ \text{DOWN } -0.30-0.10-0.10 \\ \text{LEFT } -0.30-0.10-0.10 \\ \text{RIGHT } -0.30-0.10-0.10 \end{cases}$$

$$= -0.5 + \max \begin{cases} -0.98 \\ -0.98 \\ -0.6 \\ -0.49 \end{cases} \quad =-0.5-0.5=-1.0$$

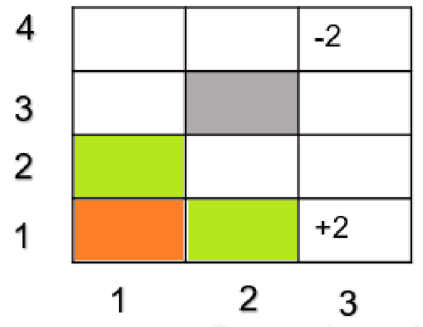
AMRITA VISHWA VIDYAPEETHAM

State values as function of iteration number



Finding policy from value function

- The action of the optimal policy is the one that maximizes the expected value function



0.7	0.4	-2
0.6	▨	0.5
0.6	0.7	0.86
0.8	0.85	+2

- For state (1,1)

$$\pi^*(1,1) = \arg\max \begin{cases} 0.6 \times V(1,2) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{UP} \\ 0.6 \times V(1,1) + 0.2 \times V(1,1) + 0.2 \times V(2,1) & \text{DOWN} \\ 0.6 \times V(1,1) + 0.2 \times V(1,2) + 0.2 \times V(1,1) & \text{LEFT} \\ 0.6 \times V(2,1) + 0.2 \times V(1,2) + 0.2 \times V(1,1) & \text{RIGHT} \end{cases}$$

- In general: $\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a) V^{\pi^*}(s')$

Value Iteration Process

		-2
	▨	
		+2

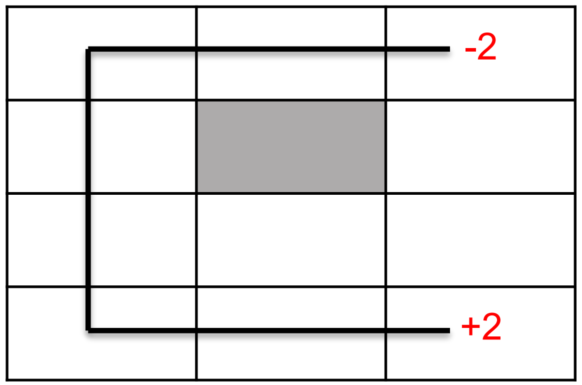
1.Given environment

0.7	0.4	-2
0.6	▨	0.5
0.6	0.7	0.86
0.8	0.85	+2

2.Calculate State Values

↓	←	-2
↓	▨	↓
→	→	↓
→	→	+2

3.Extract optimal policy



4.Execute actions

Value Iteration Algorithm

```
1. Initialize V arbitrarily V(s)=0 for all s∈$S^+$
2. Repeat
        Δ←0
        For each s∈S:
            $v \leftarrow V(s)$
            V(s)← $max_a \, _{s'} \, P^{a}_{ss'} [R^{a}_{ss'} + \gamma V(s')]$
        Δ←max(Δ,|v-V(s) |)
Until Δ<θ (a small positive number)

Output: π,such that
π(s)=argmax_a $_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V(s')]$
```

Namah Shivaya