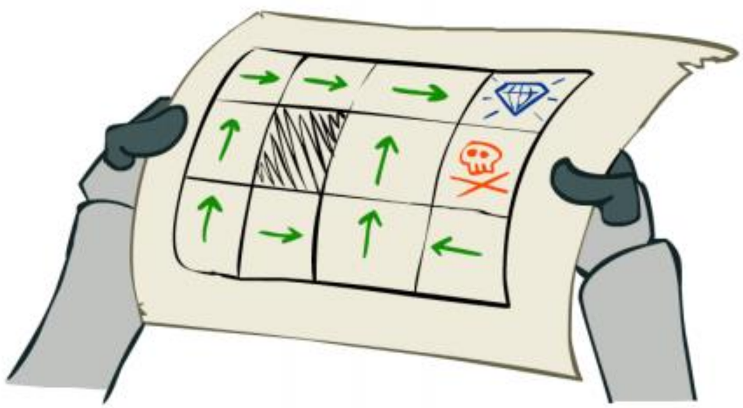
L7.1 Markov Decision Process



Zonghua Gu, Umeå University

Nov. 2023

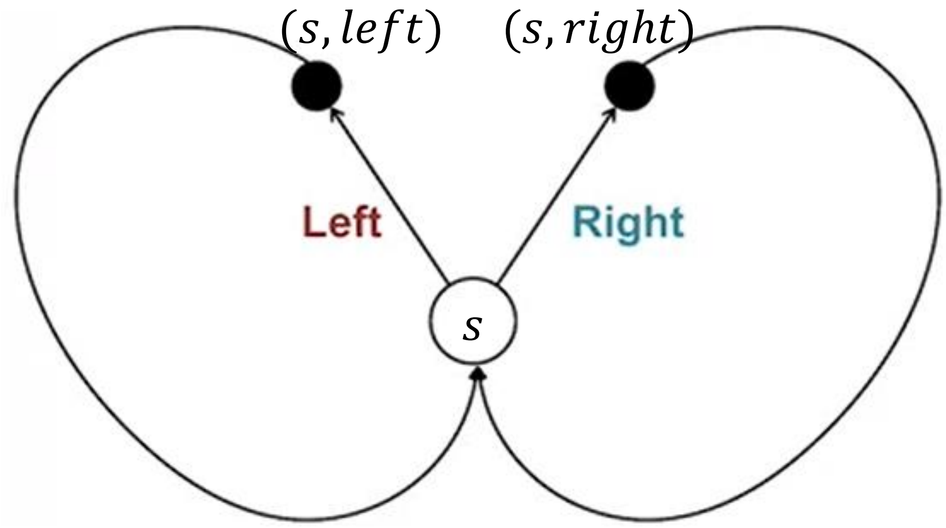
Acknowledgement: slides based on https://www.coursera.org/specializations/reinforcement-learning
And textbook by Sutton and Barto http://incompleteideas.net/book/the-book-2nd.html

Markov Decision Process (MDP)

- An MDP consists of:
  - Set of states S
  - Start state s_0
  - Set of actions A
  - Transitions and rewards $p(r, s'|s, a)$ (w. discount γ)
- Policy maps from states to actions:
  - Deterministic policy $a = \pi(s)$ defines a deterministic action a for state s.
  - Stochastic policy $\pi(a|s)$ defines a probability distribution over possible actions a for state s.
- Markov means that next state only depends on current state
  - $P\left(S_{t+1} = s' \middle| S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = a_{t-1,\dots,} S_0 = s_0, A_0 = a_0\right)$
  - $= P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$
  - Given the present state, the future and the past are independent
  - e.g., for driving task, current vehicle position $\boldsymbol{x}$ as the state does not satisfy the Markov property, since the next state depends on not only $\boldsymbol{x}$, but also velocity $\dot{\boldsymbol{x}}$, acceleration $\ddot{\boldsymbol{x}}$. (assuming acceleration $\ddot{\boldsymbol{x}}$ stays constant within each step) If we redefine the state as vector $[\boldsymbol{x}, \dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}}]^T$, then it satisfies the Markov property.
  - Or, current snapshot of front camera view can be used as the state (e.g., NVIDIA's PilotNet), but some works use past N video frames as the state to capture more dynamics (e.g., Waymo's ChauffeurNet).

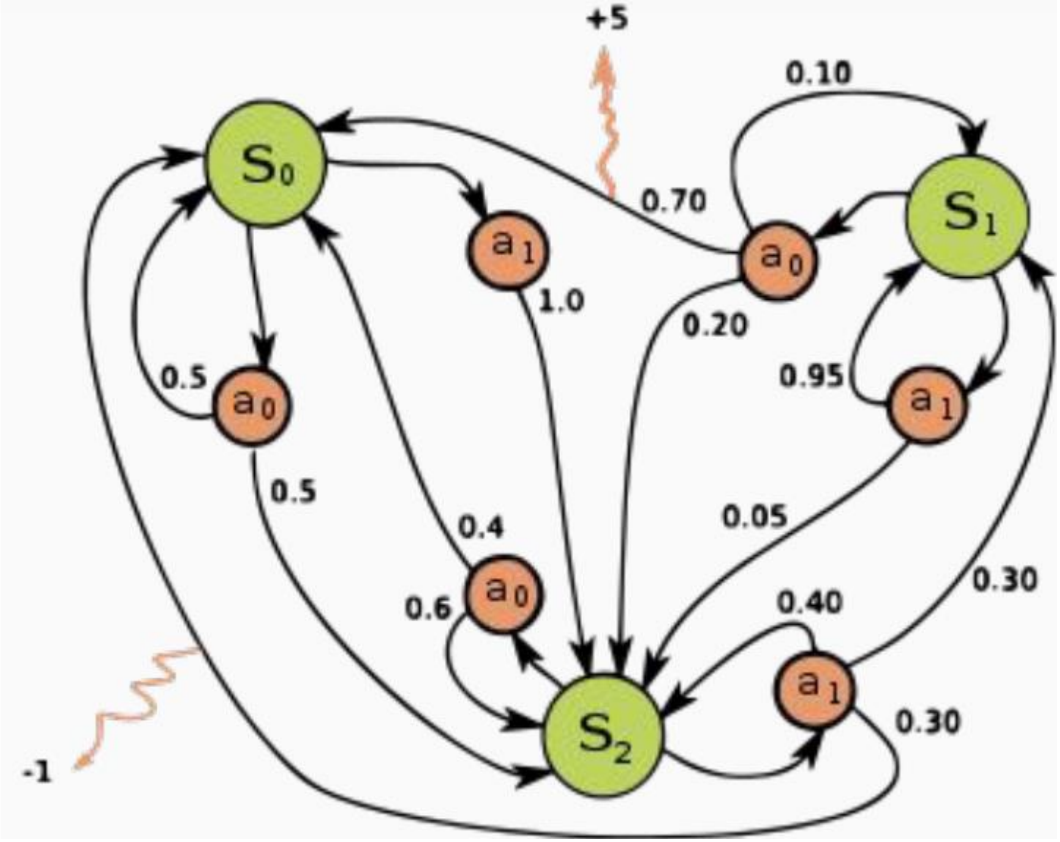
MDP Quiz

- For this MDP with a single state s and two possible actions $left$ and $right$. Are these valid policies?
  - 1) $\pi(left|s) = \pi(right|s) = 0.5$ (goes left or right with equal probability. uniform random policy)
  - 2) $\pi(left|s) = 1.0, \pi(right|s) = 0$ (always goes left)
  - 3) Alternating left and right, i.e., if previous action is left, then current action must be right, next action must be left, and so on.
  - ANS: 3) is not a valid policy, since it depends on the history of actions. To be a valid policy, the action must depend on the current state only
- We can redefine the MDP' extended state to include the last action as part of it, then 3) is a valid policy for the new MDP



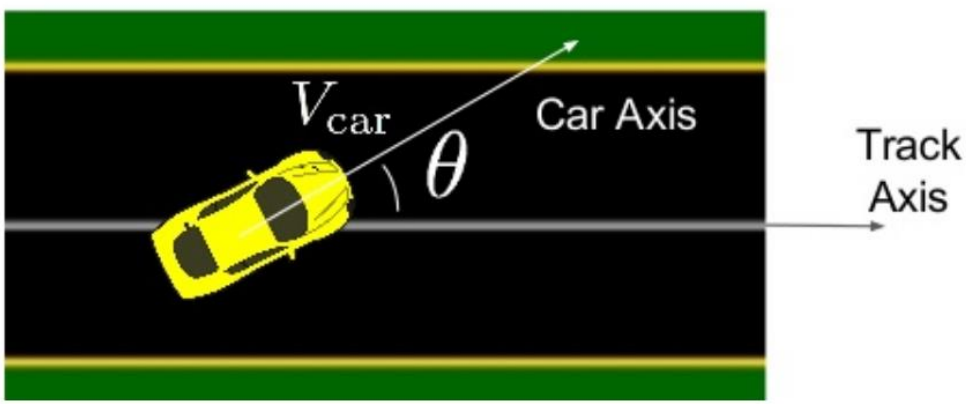
An Example MDP

- Green nodes denote 3 states s_0, s_1, s_2; Red nodes denote 2 possible actions a_0, a_1 in each state. Each red node can also be denoted as (s, a).
- Agent taking action a in state s may get different reward r and next state s', denoted as state transition (s, a, r, s'), due to environment uncertainty (all rewards are 0 expect $+5$ and -1 show in fig).



RL Reward Function

- For the vehicle in left fig:
  - state: Pose of ego-car (x, y, θ) and environment map; action: Steering wheel/brake/acceleration
- Possible reward function: $R_t = w_1 V_{car} \cos\theta - w_2|cte|$
  - Weighted sum to maximum longitudinal velocity (first term), and minimize cross-track error (distance to lane center)
  - This is an example of dense reward (e.g., at every time step), as opposed to sparse reward (e.g., only at the end of each episode)
- Compare with twiddle() :
  - twiddle() can be viewed as an RL algorithm (policy gradient), that learns PID parameters with sparse reward (cost function is average cross-track error (cte), computed at the end of each simulation episode, as sum of squares of ctes for N timesteps divided by N.
  - It does not use the numeric value of cte, only its relative size (if err < best_err);
  - Cost function does not include heading angle θ;
  - if the track is very long and irregular, then we can make the reward denser, to adjust PID parameters every K timesteps instead of at the end of each episode.



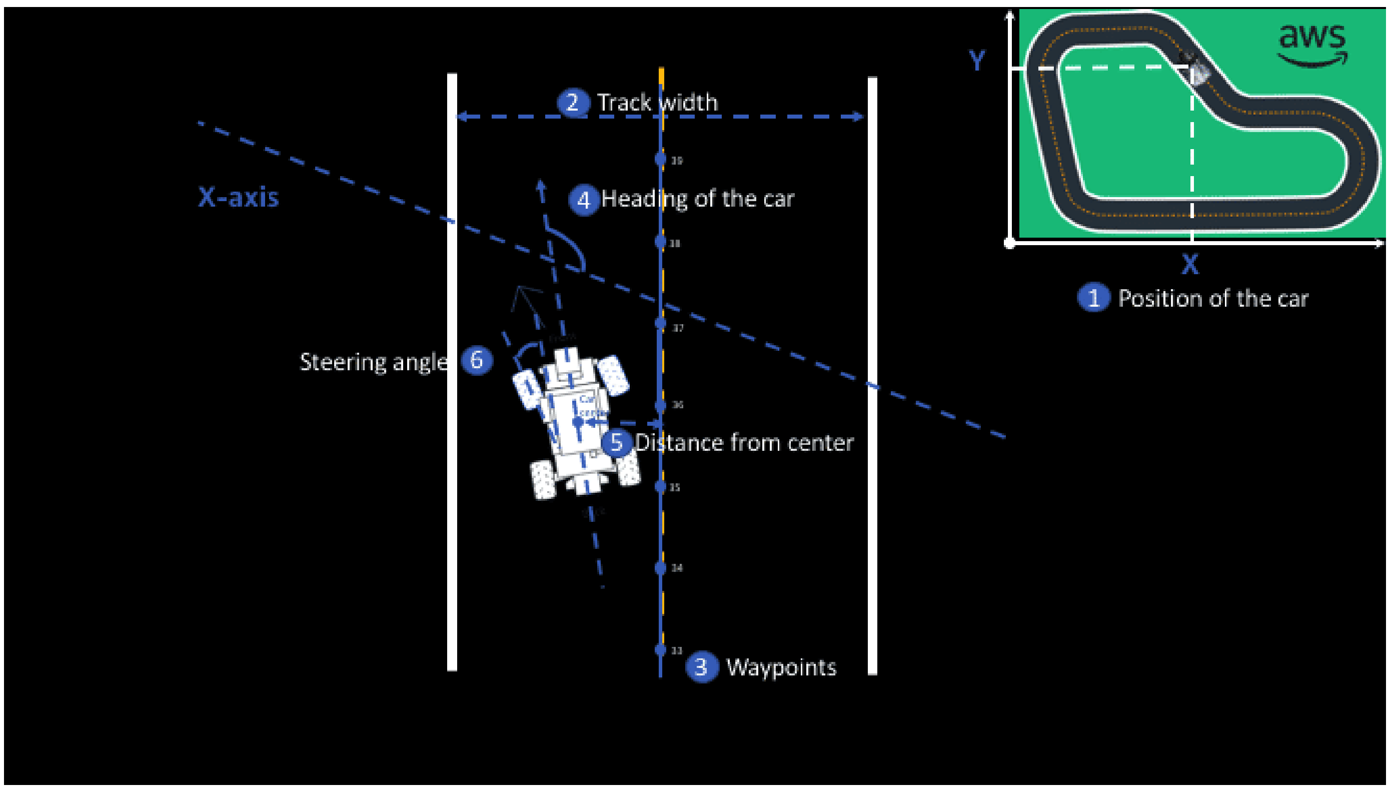

```
if err < best_err:
    best_err = err
    dp[i] *= 1.1
else:
    p[i] -= 2 * dp[i]
    robot = make_robot()
    x_trajectory, y_trajectory, err = run(robot, p)

    if err < best_err:
        best_err = err
        dp[i] *= 1.1
    else:
        p[i] += dp[i]
        dp[i] *= 0.9
```

twiddle()

Amazon DeepRacer

- Amazon Web Services (AWS) launched DeepRacer in 2018 for training AD algorithms with RL
  - https://aws.amazon.com/deepracer/
- You can train RL algorithm in the simulator on AWS cloud, but it costs money after some free time.
- They hold competitions, both online and in real-world. 1/10th scale race car costs USD $349.

Params for Writing Reward Function



- all_wheels_on_track
- x
- y
- distance_from_center
- is_left_of_center
- is_reversed
- heading
- progress
- steps
- speed
- steering_angle
- track_width
- waypoints
- closest_waypoints

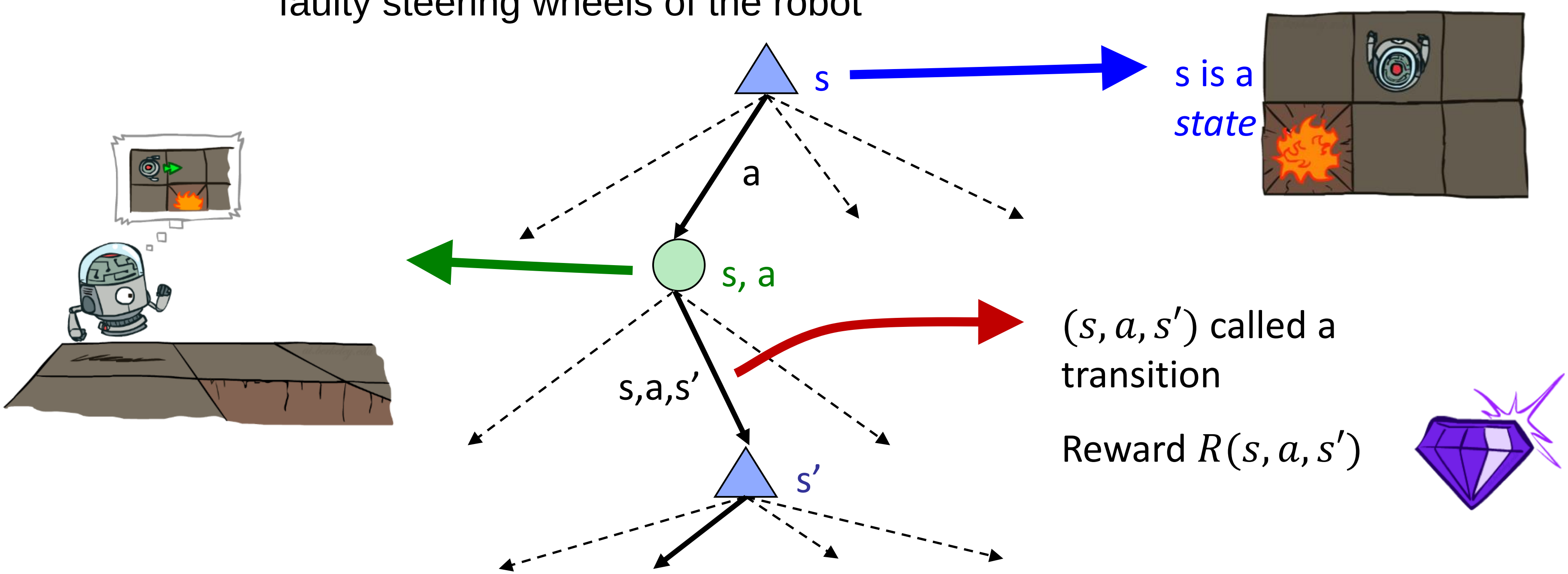
Example Reward Function

- def reward_function(params):
-     '''Example of penalize steering, which helps mitigate zig-zag behaviors'''
-     # Read input parameters
-     distance_from_center = params['distance_from_center']
-     track_width = params['track_width']
-     steering = abs(params['steering_angle']) # Only need the absolute steering angle
-
-     # Calculate 3 markers that are at varying distances away from the center line
-     marker_1 = 0.1 * track_width
-     marker_2 = 0.25 * track_width
-     marker_3 = 0.5 * track_width
-
-     # Give higher reward if the agent is closer to center line and vice versa
-     if distance_from_center <= marker_1:
-         reward = 1
-     elif distance_from_center <= marker_2:
-         reward = 0.5
-     elif distance_from_center <= marker_3:
-         reward = 0.1
-     else:
-         reward = 1e-3 # likely crashed/ close to off track
-
-     # Steering penalty threshold, change the number based on your action space setting
-     ABS_STEERING_THRESHOLD = 15
-
-     # Penalize reward if the agent is steering too much
-     if steering > ABS_STEERING_THRESHOLD:
-         reward *= 0.8
-
-     return float(reward)

- A more realistic and complex reward function: https://www.middleware-solutions.fr/2019/08/14/an-introduction-to-aws-deepracer

MDP Search Tree

- Each MDP state s projects a search tree starting from it
- Both policy and environment may be stochastic
  - Policy $\pi(a|s)$: probability distribution over possible actions a from state s
    - e.g., different actions may be taken for the same state
  - Environment $p(r, s'|s, a)$: if the agent takes action a in state s, env gives probability distribution over next state s' and reward r
    - e.g., due to non-determinism in the environment (sudden strong wind), or faulty steering wheels of the robot

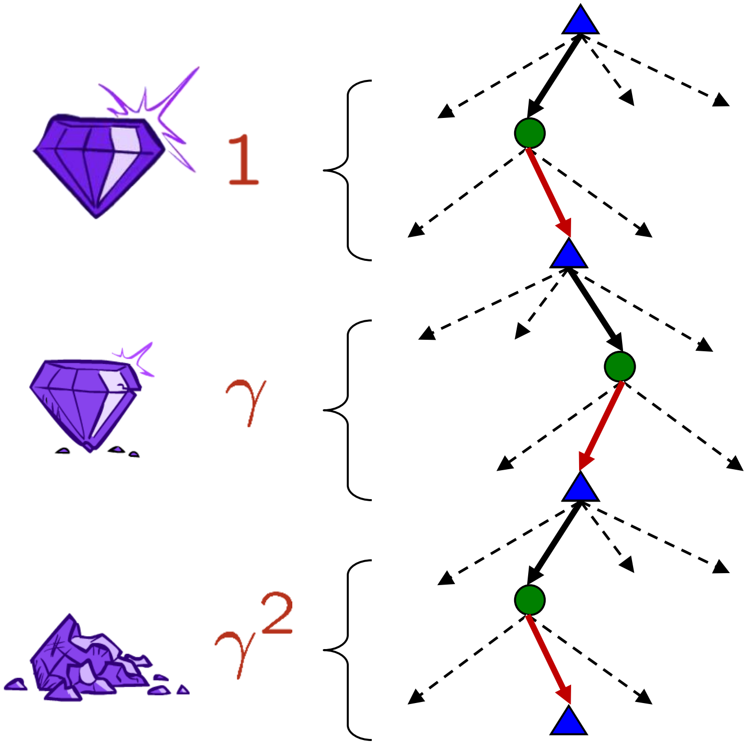


Preventing Infinite Rewards

- Problem: What if the game lasts forever? Do we get infinite rewards? No. Possible solutions:
- **Finite horizon**: (limit search tree depth)
  - Terminate episodes after a fixed T timesteps
- **Discount factor**: $0 < \gamma \leq 1$
  - Think of it as a $1 - \gamma$ chance of ending the episode at every step. Effective horizon (expected episode length): $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$
    - $\sum_{t=0}^{\infty} .1^t = \frac{1}{1-.1} = 1.1, \sum_{t=0}^{\infty} .5^t = \frac{1}{1-.5} = 2, \sum_{t=0}^{\infty} .9^t = \frac{1}{1-.9} = 10$
  - Smaller γ leads to shorter horizon, and preference of short-term to long-term rewards, and vice versa
- (Can have both finite horizon and discount factor)

Discount Factor Example

- Each time we descend a level in the search tree, we multiply in the discount once
- Example: $\gamma = 0.5$
  - $U([1,2,3]) = 1 * 1 + 0.5 * 2 + 0.25 * 3 < U([3,2,1]) = 1 * 3 + 0.5 * 2 + 0.25 * 1$



Discounting Example

- Given:
  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic
- For $\gamma = 1$, optimal policy in each state is always moving West
  - From state d, reward of going West is $0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 10 = 10$, larger than reward of going East $0 + \gamma \cdot 1 = 1$
- For $\gamma = 0.1$, optimal policy in each state is shown below
  - From state d, reward from going West is $0 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \gamma^3 \cdot 10 = 0.01$, less than reward from going East $0 + \gamma \cdot 1 = 0.1$.
- For which γ are West and East equally good when in state d?
  - $\gamma^3 \cdot 10 = \gamma \cdot 1 \Longrightarrow \gamma = \frac{1}{\sqrt{10}} \approx .32$

10	←	←	←	1
a	b	c	d	e

$\gamma = 1$

10	←	←	→	1
a	b	c	d	e

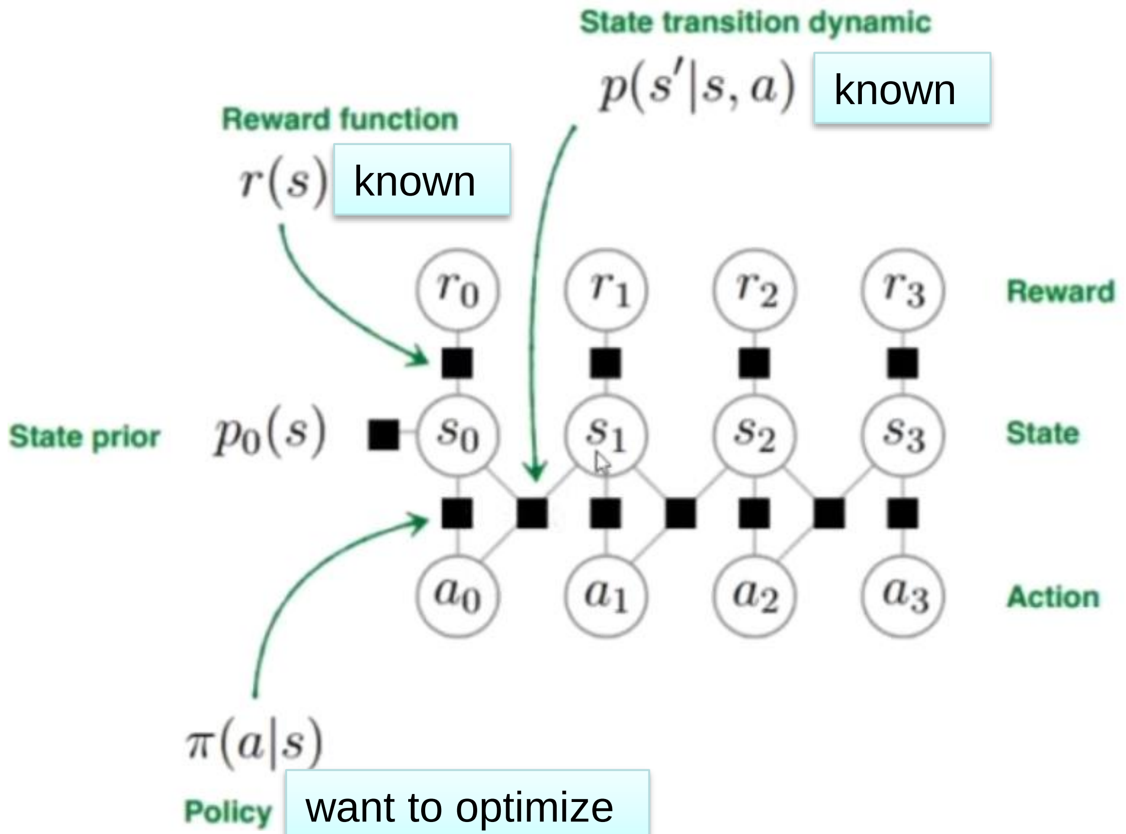
$\gamma = 0.1$

10	←	←	⇆	1
a	b	c	d	e

$\gamma = \frac{1}{\sqrt{10}}$

Known MDP

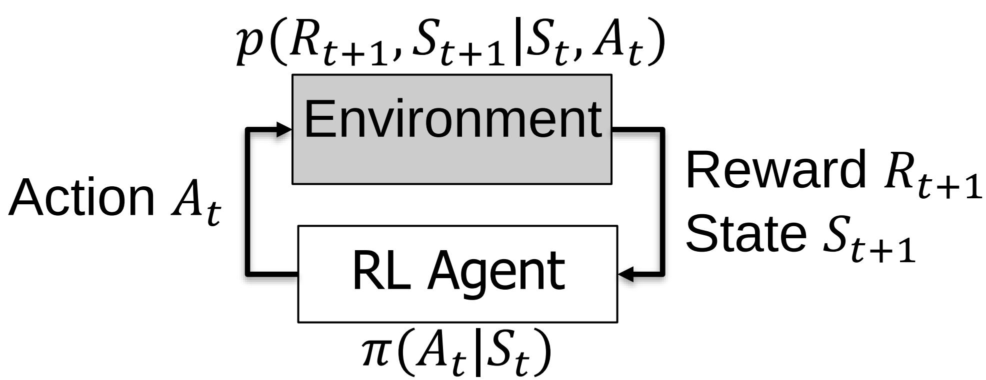
- In this lecture, we assume known MDP, and use dynamic programming to solve Bellman Equations and find the optimal policy (no learning here).



Important

Formal Definition of MDP

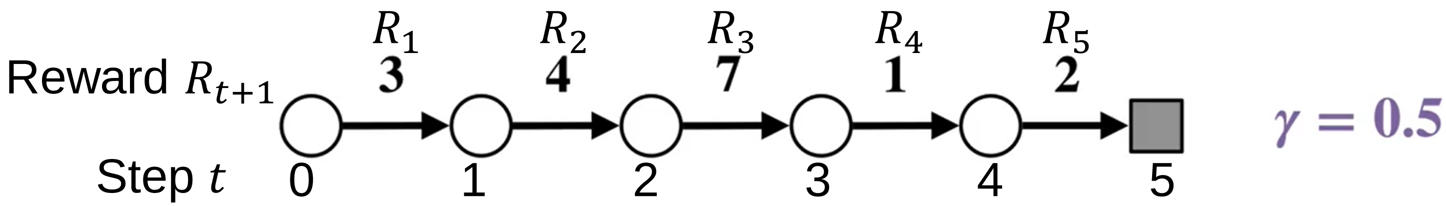
- Return (cumulative discounted reward) at time t: $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$
  - At each step $t \in [0, T-1]$, agent takes an action A_t in state S_t; at step $t+1$, agent receives a reward R_{t+1} and transitions into the next state S_{t+1} with the trace $(S_t, A_t, R_{t+1}, S_{t+1})$
  - We assume episodic tasks, and this specific episode has length of T steps. ($T = \infty$ for continuous tasks)
- State Value Function: expected return under policy π: $v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$
- Action Value Function: expected return from taking action a, then follow policy π: $q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$
- The RL problem: find the optimal policy $\pi(a|s)$ that maximizes the expected return from each state



Important

Example: Computing Returns for One Episode

- Working backward is more efficient than working forward as it avoids redundant computations.



$$G_0 = R_1 + \gamma G_1$$
$$G_1 = R_2 + \gamma G_2$$
$$G_2 = R_3 + \gamma G_3$$
$$G_3 = R_4 + \gamma G_4$$
$$G_4 = R_5 + \gamma G_5$$
$$G_5 = 0$$

$$G_0 = R_1 + \gamma \ R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7$$
$$G_1 = R_2 + \gamma \ R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8$$
$$G_2 = R_3 + \gamma \ R_4 + \gamma^2 R_5 = 8$$
$$G_3 = R_4 + \gamma \ R_5 = 2$$
$$G_4 = R_5 = 2$$
$$G_5 = 0$$

Important

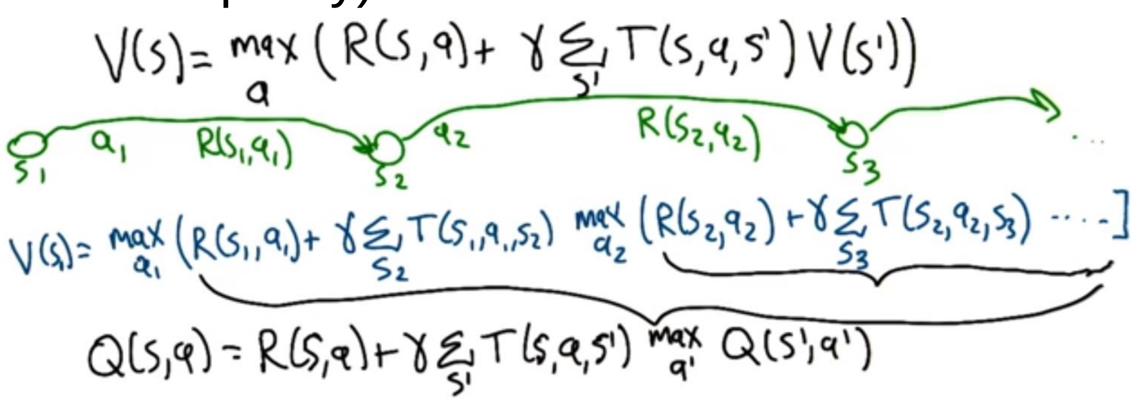
Bellman Expectation Equations

- Bellman Expectation Equation (BEE) for State Value Function:
- $v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) \, [r + \gamma v_\pi(s')]$
  - Expected value starting from state s and following policy π.
- Bellman Expectation Equation for Action Value Function
- $q_\pi(s, a) = \sum_{r,s'} p(r, s'|s, a) \, [r + \gamma \sum_{a'} \pi(a'|s') \, q_\pi(s', a')]$
  - Expected value starting from state s, taking action a, and thereafter following policy π.

Important

Bellman Optimality Equations

- Bellman Optimality Equation (BEE) for the Optimal State Value Function:
- $v_*(s) = \max_{a} \sum_{r,s'} p(r,s'|s,a)\,[r + \gamma v_*(s')]$
  - Max value starting from state s and following the greedy policy $\pi(s) = \operatorname{argmax}_{a} q_*(s,a)$ (the optimal policy)
- Bellman Optimality Equation for the Optimal Action Value Function
- $q_*(s,a) = \sum_{r,s'} p(r,s'|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$
  - Max value starting from state s, taking action a, and thereafter following the greedy policy $\pi(s) = \operatorname{argmax}_{a} q_*(s,a)$ (the optimal policy)



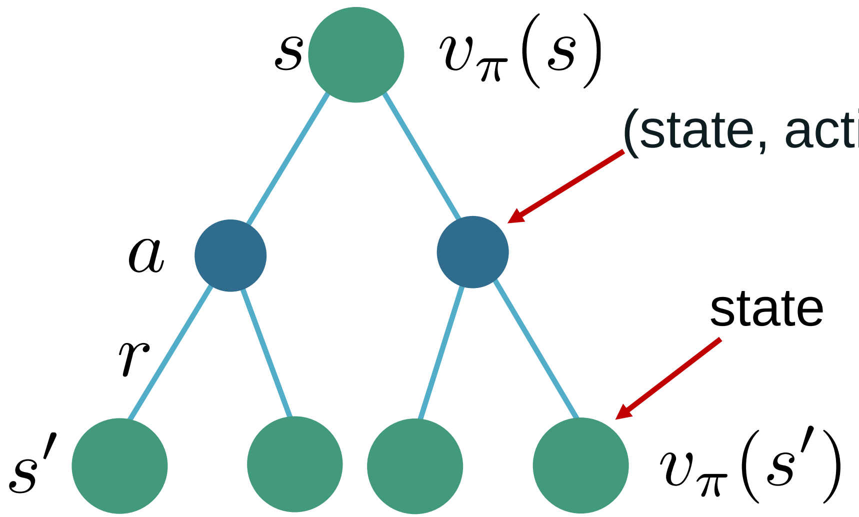
- Notations in left fig:
- $\sum_{s'} T(s,a,s')\,[\dots] = \sum_{r,s'} p(r,s'|s,a)\,[\dots]$
- $R(s,a) = \sum_{r,s'} p(r,s'|s,a)\, r$

Bellman Equations written with Expectation Symbols

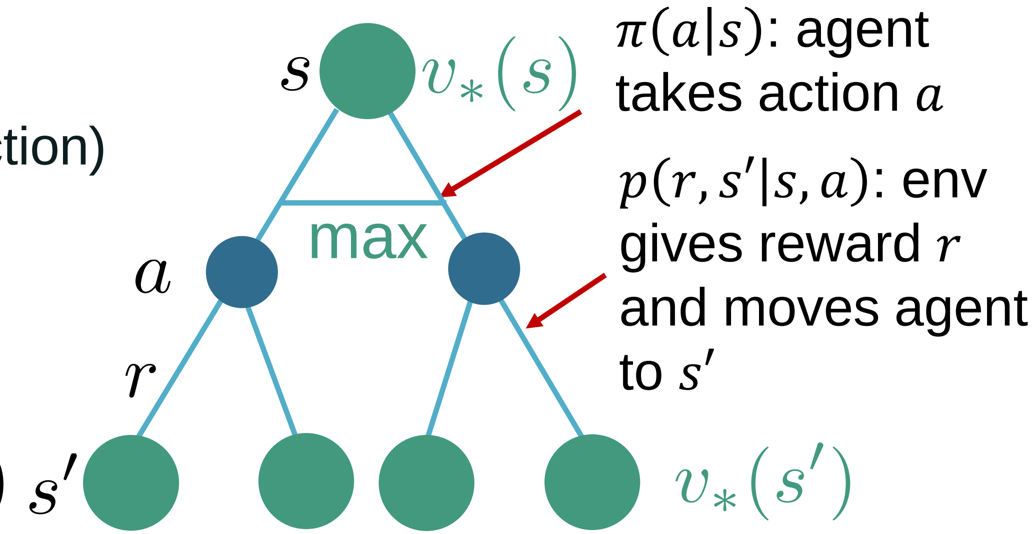
- BEE:
- $v_\pi(s) = \mathbb{E}_a \mathbb{E}_{r,\, s'}[r + \gamma v_\pi(s')]$
- $q_\pi(s,a) = \mathbb{E}_{r,\, s'}[r + \gamma \mathbb{E}_a q_\pi(s,a)]$
- BOE:
- $v_*(s) = \max_a \mathbb{E}_{r,\, s'}[r + \gamma v_*(s')]$
- $q_*(s,a) = \mathbb{E}_{r,\, s'}\left[r + \gamma \max_a q_*(s,a)\right]$
- Detailed derivations:
  - $v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a) = \mathbb{E}_{a\sim\pi(a|s)} q_\pi(s,a) = \mathbb{E}_{a\sim\pi(a|s)}\, \mathbb{E}_{r,\, s'\sim p(r,s'|s,a)}[r + \gamma v_\pi(s')]$
  - $q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a)\, [r + \gamma v_\pi(s')] = \mathbb{E}_{r,\, s'\sim p(r,s'|s,a)}[r + \gamma v_\pi(s')] = \mathbb{E}_{r,\, s'\sim p(r,s'|s,a)}\left[r + \gamma \mathbb{E}_{a\sim\pi(a|s)}\, q_\pi(s,a)\right]$
  - $v_*(s) = \max_a q_*(s,a) = \max_a \mathbb{E}_{r,\, s'\sim p(r,s'|s,a)}[r + \gamma v_*(s')]$
  - $q_*(s,a) = \sum_{r,s'} p(r,s'|s,a)\, [r + \gamma v_*(s')] = \mathbb{E}_{r,\, s'\sim p(r,s'|s,a)}[r + \gamma v_*(s')]$

Important

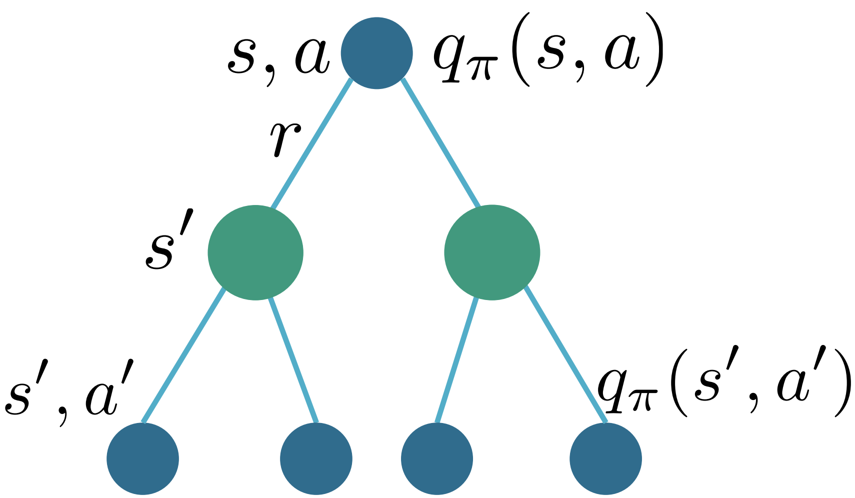
Backup Diagrams



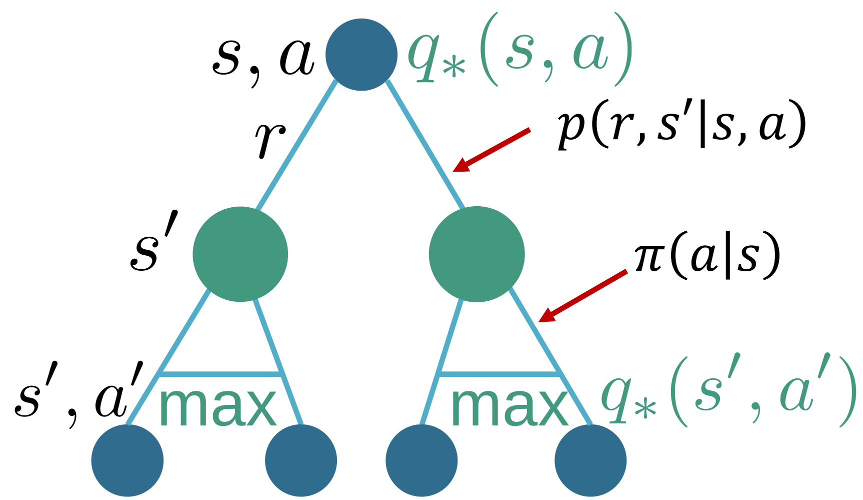
Bellman Exp Eqn for $v_\pi(s)$



Bellman Opt Eqn for $v_*(s)$



Bellman Exp Eqn for $q_\pi(s, a)$



Bellman Opt Eqn for $q_*(s, a)$

$v(s)$ vs. $q(s,a)$

- State-action Value Function $q(s,a)$ contains more information than State value function $v(s)$. Given $q_*(s,a)$, optimal policy $\pi_*(\mathrm{s}) = \operatorname{argmax}_a q_*(s,a)$.
- Can always go from $q_\pi(s,a)$ to $v_\pi(s)$, or from $q_*(s,a)$ to $v_*(s)$:
  - $v_\pi(s) = \sum_a \pi(a|s)\, q_\pi(s,a)$; $v_*(s) = \max_a q_*(s,a)$
- With known MDP ($p(r,s'|s,a)$, i.e., model-based): can go from $v_\pi(s)$ to $q_\pi(s,a)$, or from $v_*(s)$ to $q_*(s,a)$:
  - $q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a)\,[r + \gamma v_\pi(s')]$
  - $q_*(s,a) = \sum_{r,s'} p(r,s'|s,a)\,[r + \gamma v_*(s')]$
- With unknown MDP (unknown $p(r,s'|s,a)$, i.e., model-free) : cannot go from $v_\pi(s)$ to $q_\pi(s,a)$, or from $v_*(s)$ to $q_*(s,a)$

Simplified Bellman Equations for Deterministic Env

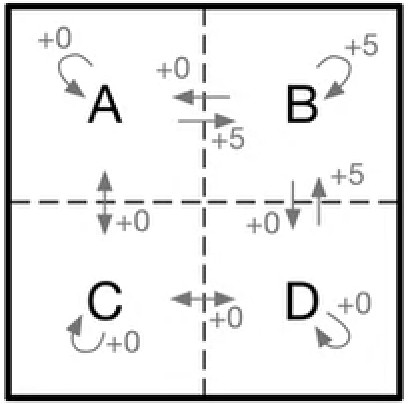
- Bellman Equations:
  - $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$
  - $q_\pi(s,a) = \sum_{r,s'} p(r,s'|s,a)\,[r + \gamma v_\pi(s')]$
  - $v_*(s) = \max_a q_*(s,a)$
  - $q_*(s,a) = \sum_{r,s'} p(r,s'|s,a)\,[r + \gamma v_*(s')]$
- For Deterministic Env: there is only one possible (r,s') for a given (s,a) (we use R_s^a to emphasize that reward r is specific to this (s,a)):
  - $q_\pi(s,a) = R_s^a + \gamma v_\pi(s')$
  - $q_*(s,a) = R_s^a + \gamma v_*(s')$

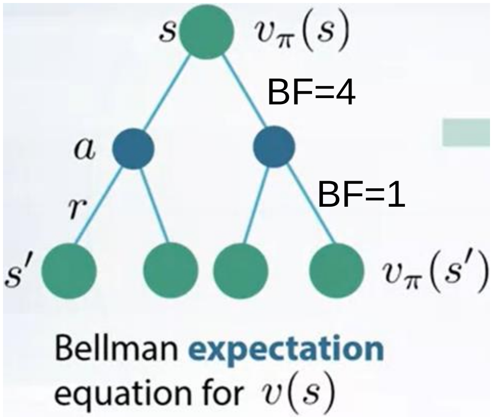
Policy Evaluation

- The prediction problem: predict Value Function for given policy π by solving Bellman Exp. Equation for State Value Function
  - $v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$
    $= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$
- Can also be written as:
  - $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$
  - $q_\pi(s, a) = \sum_{r,s'} p(r, s'|s, a) [r + \gamma v_\pi(s')]$ denotes the State-Action Value Function for taking action a in state s, then follow policy π afterwards
- A set of linear equations that can be solved analytically for small system
  - # unknowns = # equations = # states

Grid World1: Policy Evaluation

- Non-episodic MDP w. deterministic env: Agent in state $s \in \{A, B, C, D\}$ taking action $a \in \{l, r, u, d\}$ always moves to the next state in the movement direction, unless it is blocked by the walls. Discount factor $\gamma = 0.7$.
- Random policy: Agent in state $s \in \{A, B, C, D\}$ takes a random action $a \in \{l, r, u, d\}$ with equal probability of 0.25 each.
- Bellman Exp. Equation for det env: $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)\,;\; q_\pi(s,a) = R_s^a + \gamma v_\pi(s')$
- $v_\pi(A) = 0.25(q_\pi(A,l) + q_\pi(A,r) + q_\pi(A,u) + q_\pi(A,d)) = 0.5 \cdot 0.7 v_\pi(A) + 0.25 \cdot \big(5 + 0.7 v_\pi(B)\big) + 0.25 \cdot 0.7 v_\pi(C)$
  - $q_\pi(A,l) = q_\pi(A,u) = 0 + 0.7 v_\pi(A)$
  - $q_\pi(A,r) = 5 + 0.7 v_\pi(B)$
  - $q_\pi(A,d) = 0 + 0.7 v_\pi(C)$
- $v_\pi(B) = 0.25(q_\pi(B,l) + q_\pi(B,r) + q_\pi(B,u) + q_\pi(B,d)) = 0.25 \cdot 0.7 v_\pi(A) + 0.5 \cdot \big(5 + 0.7 v_\pi(B)\big) + 0.25 \cdot 0.7 v_\pi(D)$
  - $q_\pi(B,l) = 0 + 0.7 v_\pi(A)$
  - $q_\pi(B,r) = q_\pi(A,u) = 5 + 0.7 v_\pi(B)$
  - $q_\pi(B,d) = 0 + 0.7 v_\pi(D)$
- $v_\pi(C) = 0.25(q_\pi(C,l) + q_\pi(C,r) + q_\pi(C,u) + q_\pi(C,d)) = 0.25 \cdot 0.7 v_\pi(A) + 0.5 \cdot 0.7 v_\pi(C) + 0.25 \cdot 0.7 v_\pi(D)$
  - $q_\pi(C,l) = q_\pi(C,d) = 0 + 0.7 v_\pi(C)$
  - $q_\pi(C,r) = 0 + 0.7 v_\pi(D)$
  - $q_\pi(C,u) = 0 + 0.7 v_\pi(A)$
- $v_\pi(D) = 0.25(q_\pi(D,l) + q_\pi(D,r) + q_\pi(D,u) + q_\pi(D,d)) = 0.25 \cdot (5 + 0.7 v_\pi(B)) + 0.25 \cdot 0.7 v_\pi(C) + 0.5 \cdot 0.7 v_\pi(D)$
  - $q_\pi(D,l) = 0 + 0.7 v_\pi(C)$
  - $q_\pi(D,r) = q_\pi(D,d) = 0 + 0.7 v_\pi(D)$
  - $q_\pi(D,u) = 5 + 0.7 v_\pi(B)$
- Solution: $v_\pi(A) = 4.2, v_\pi(B) = 6.1, v_\pi(C) = 2.2, v_\pi(D) = 4.2$. $q_\pi(s,a)$ can also be obtained.





(BF: Branching Factor)