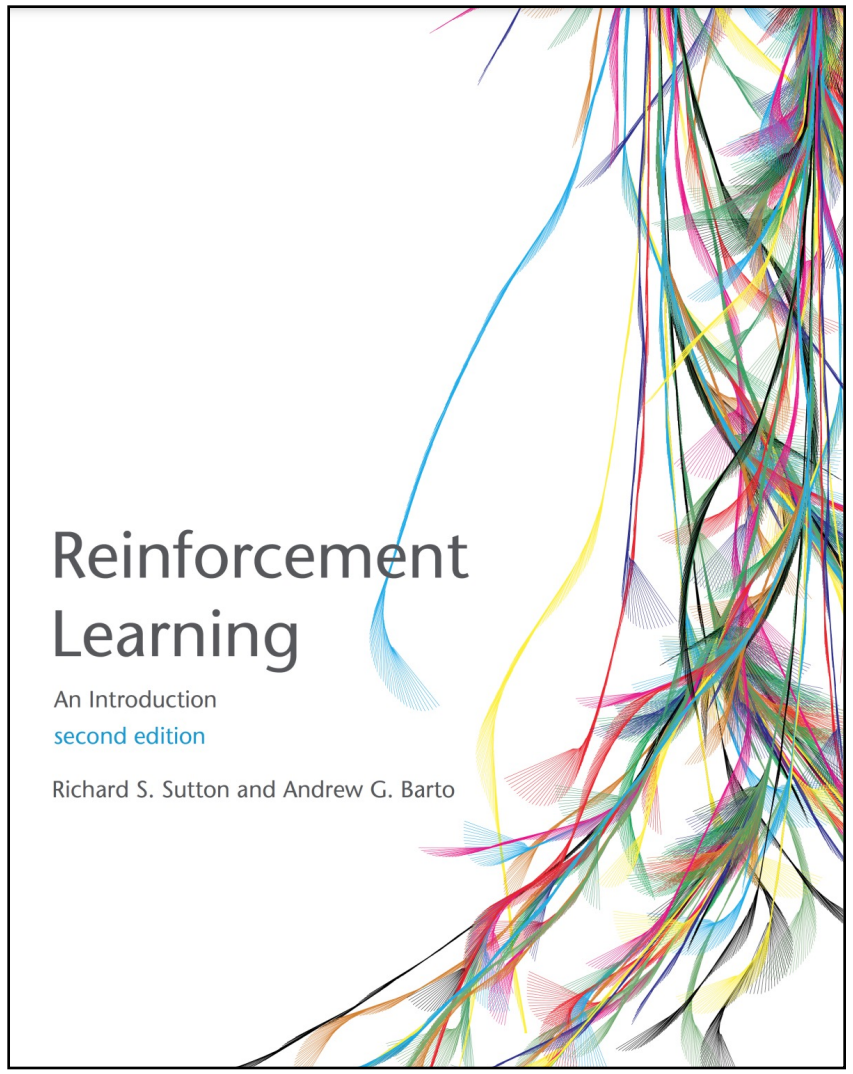
Session 2

Reinforcement Learning

KAUSHIK LAKSHMINARASIMHAN

Bibliography



02/22: Chapters 3-4

- Markov Decision Process
- Model-based Methods (*planning*)

03/01: Chapters 5-8

- Model-Free Methods (*learning*)
- RL in Neuroscience

Session 1 Recap

Markov decision process

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\},$$

Policy

$$\pi(a|s) \doteq \Pr\{A_t = a|S_t = s\}$$

Value function

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Bellman expectation equation

used for
Policy evaluation
a.k.a. *Prediction*

Optimal value function

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$$

Bellman optimality equation

used for
Policy optimization
a.k.a. *Control*

Solving Bellman optimality eqn.

Model-based methods
Dynamic programming (*Policy iteration* and *value iteration*)

Value Iteration Recap

Random initialization:
$V(s) \in \mathbb{R}$

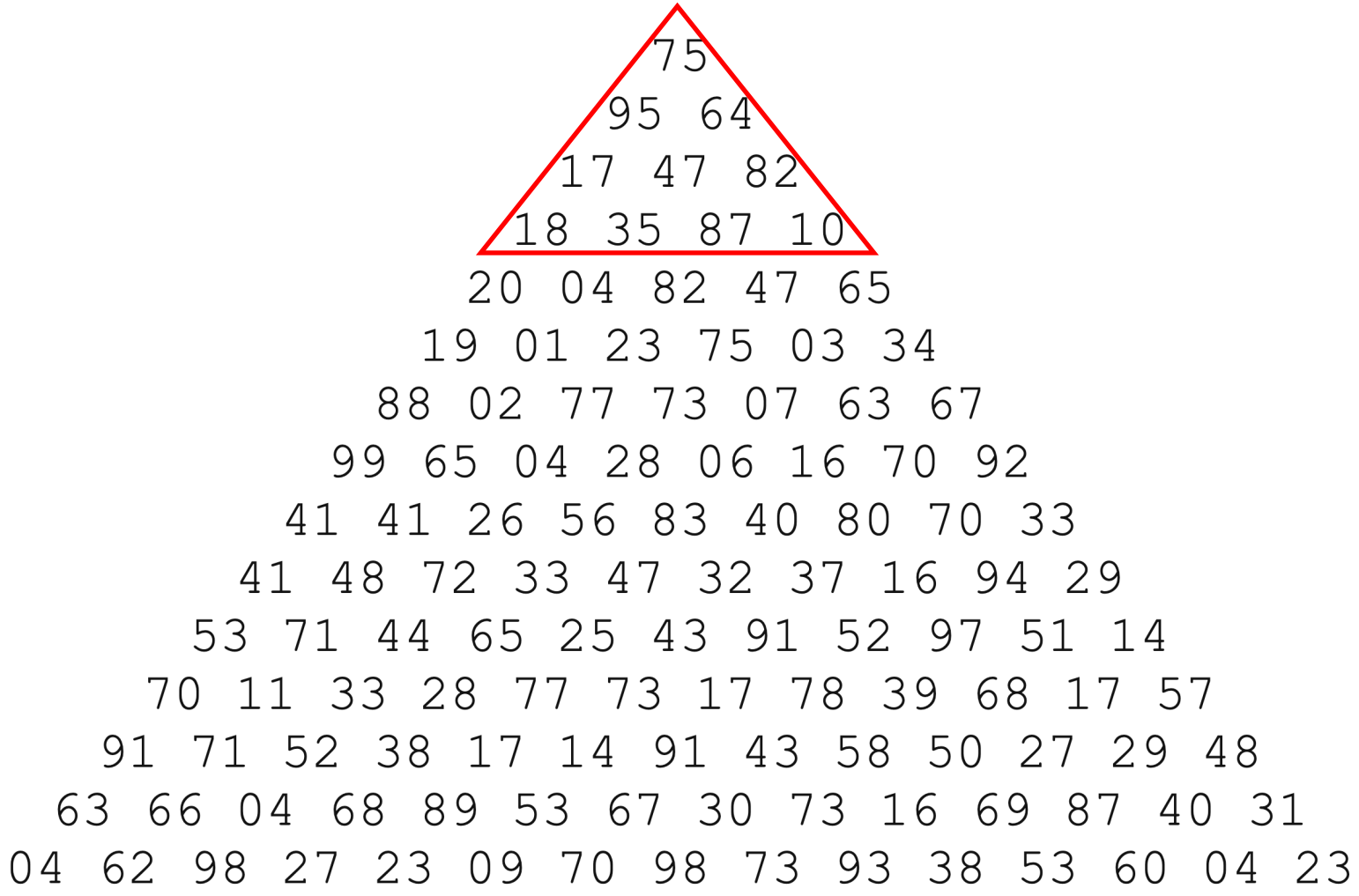
Loop until $V(s)$ is stable (combines *evaluation* and *improvement*):

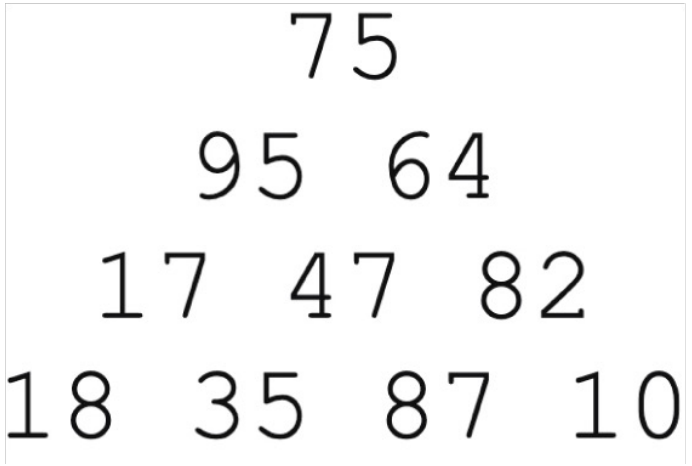
$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \text{ for each } s \in \mathcal{S}$$

$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$$

Value Iteration Recap

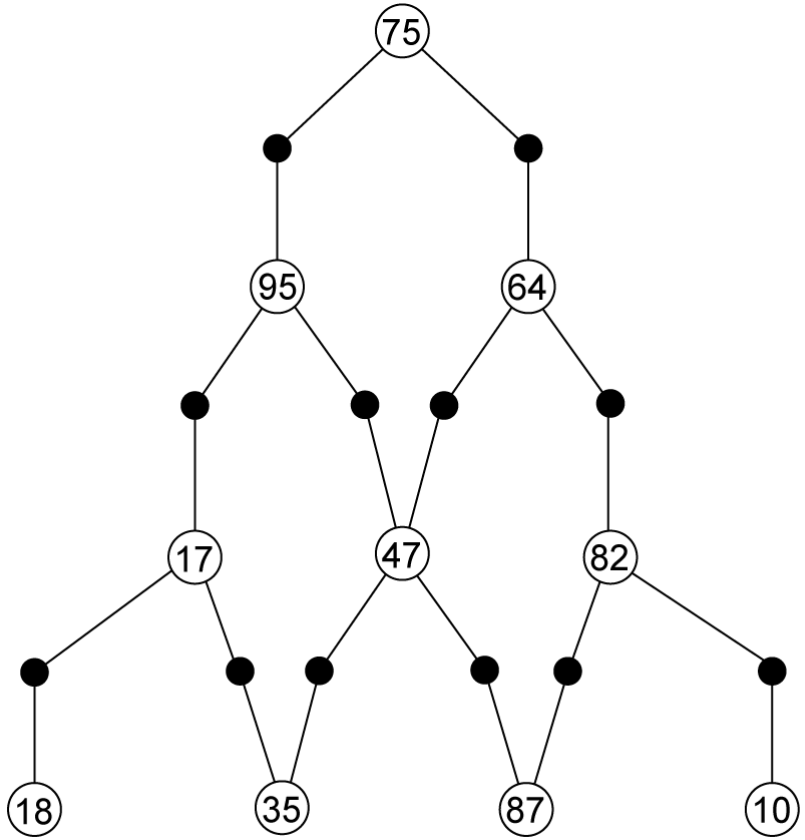
Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right





Value Iteration Recap

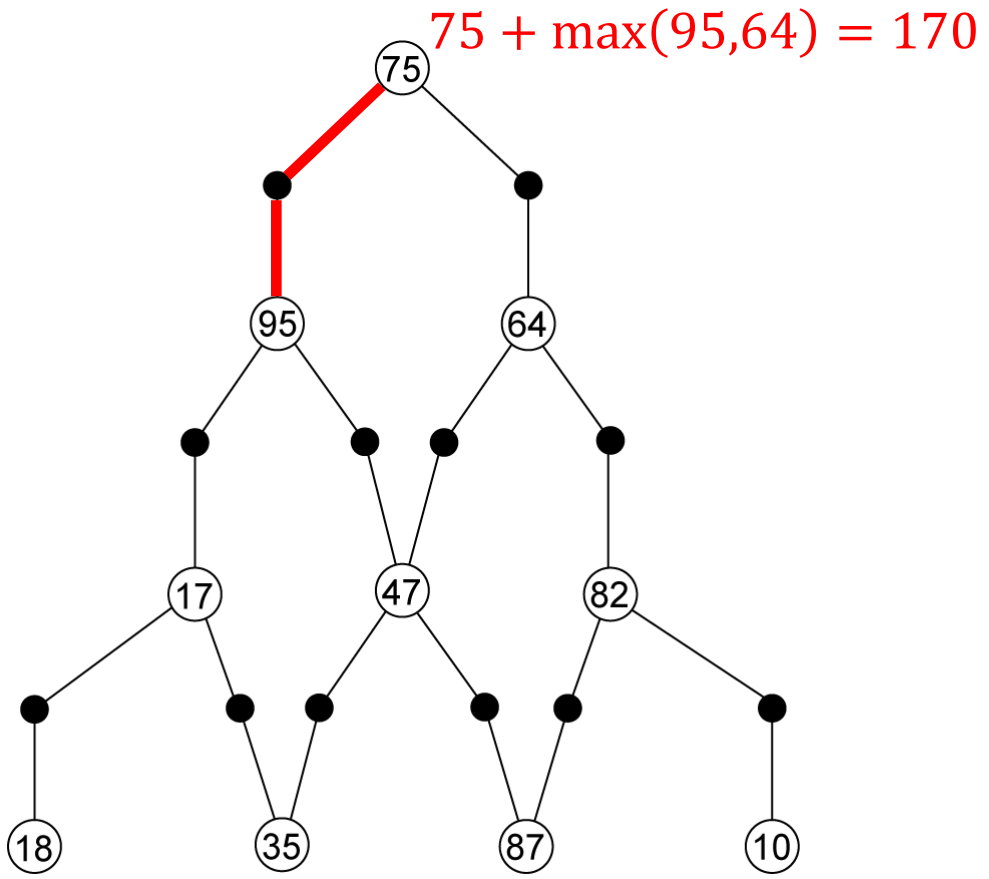
Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right



Value Iteration Recap

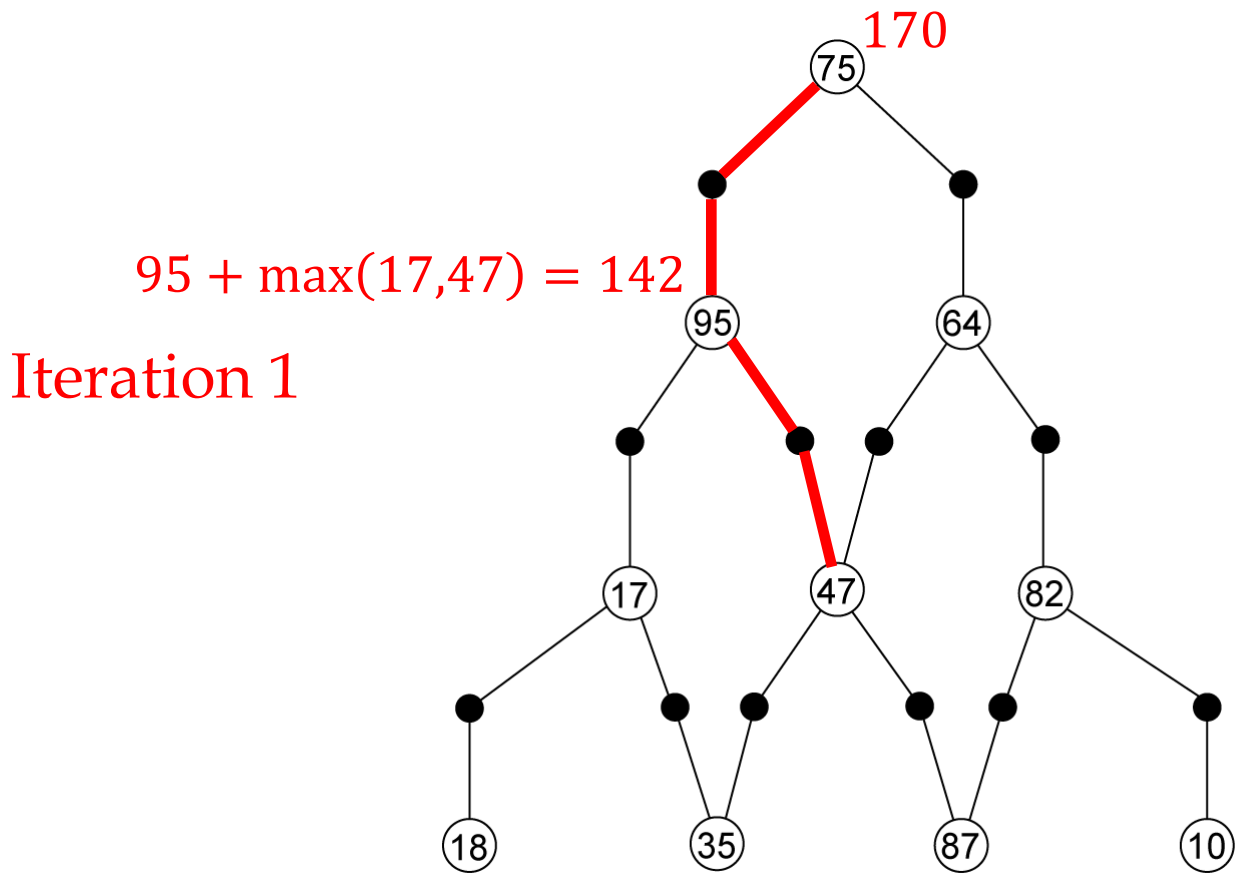
Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

Iteration 1



Value Iteration Recap

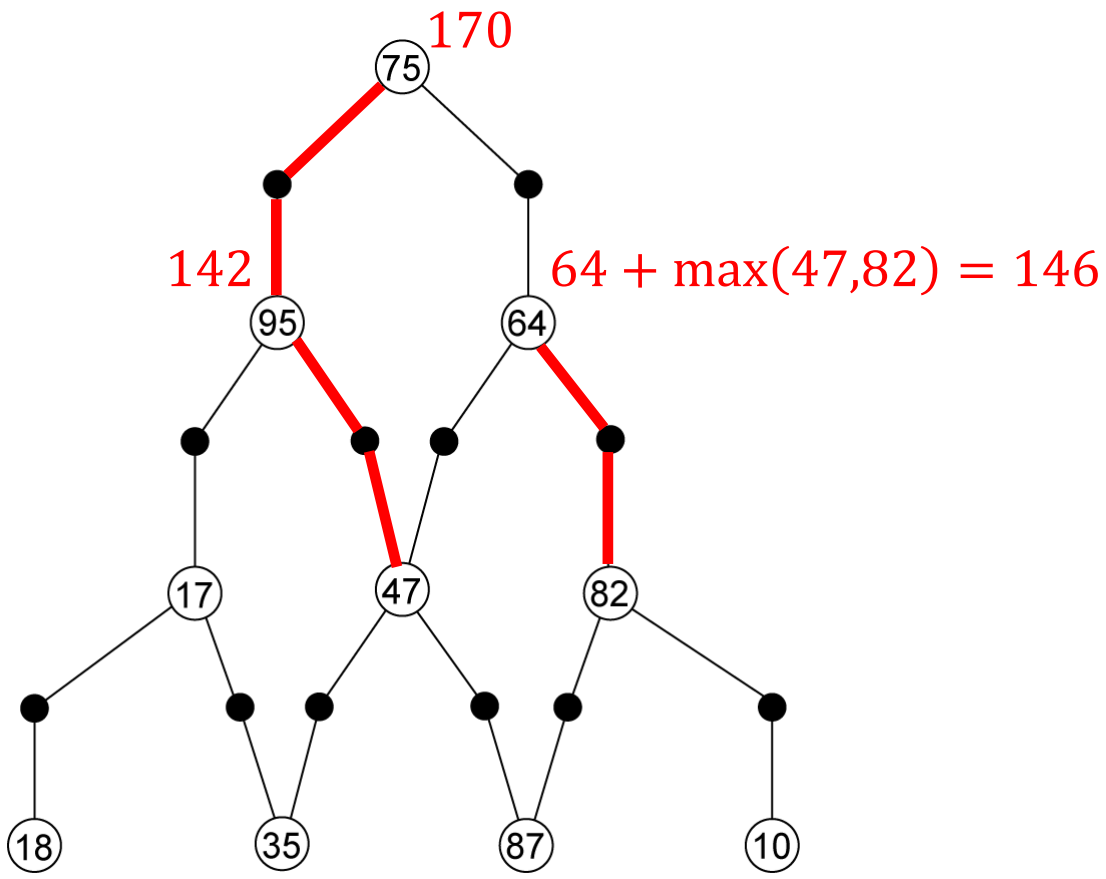
Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

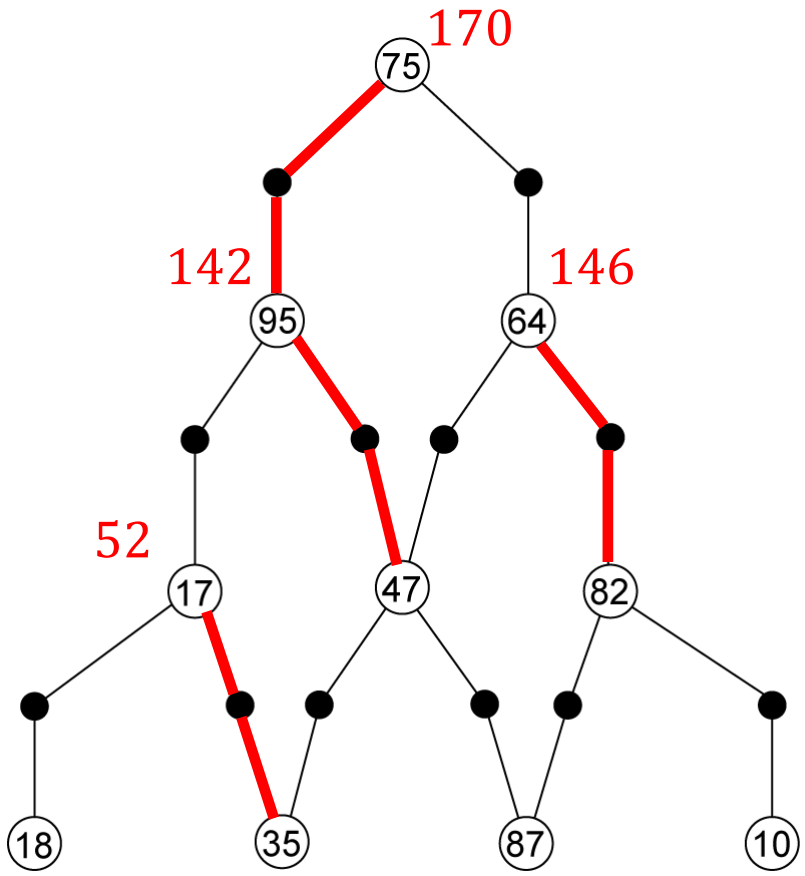
Iteration 1



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

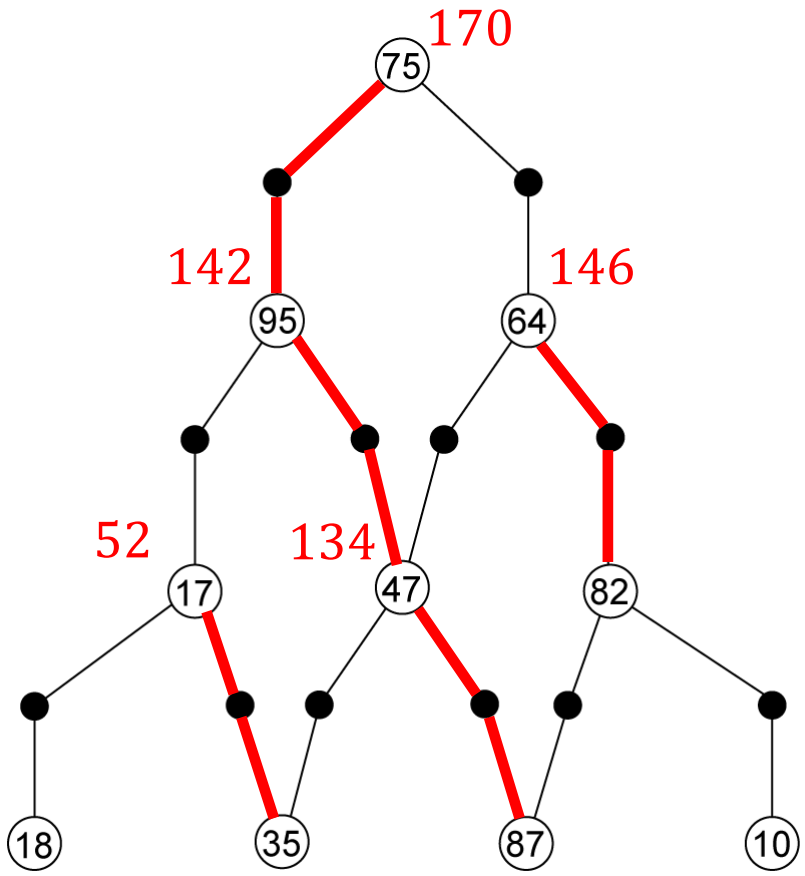
Iteration 1



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

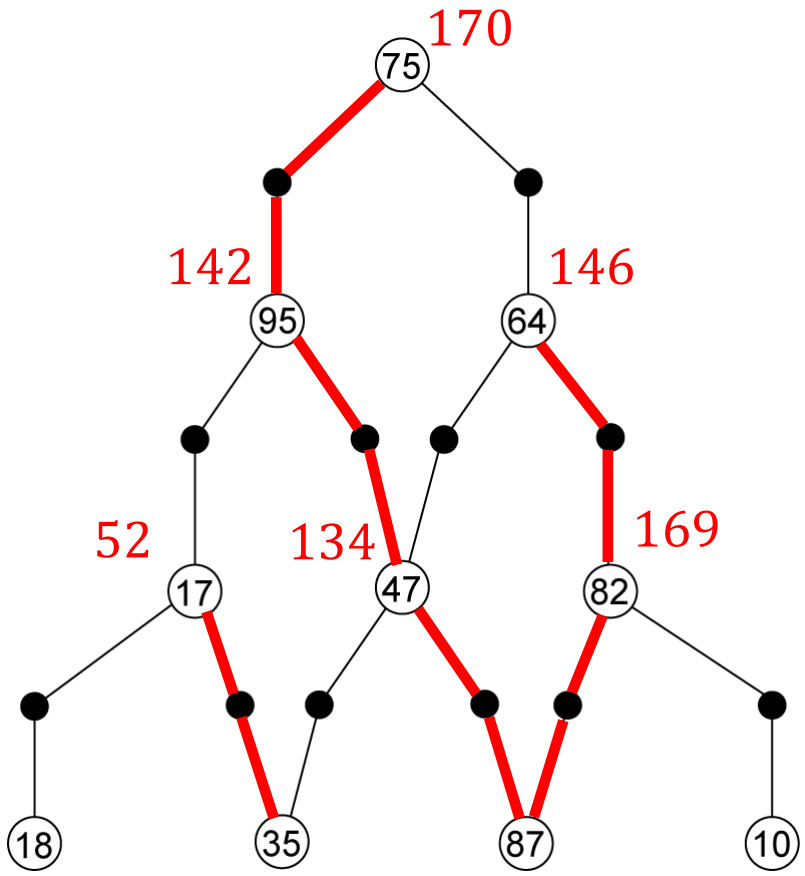
Iteration 1



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

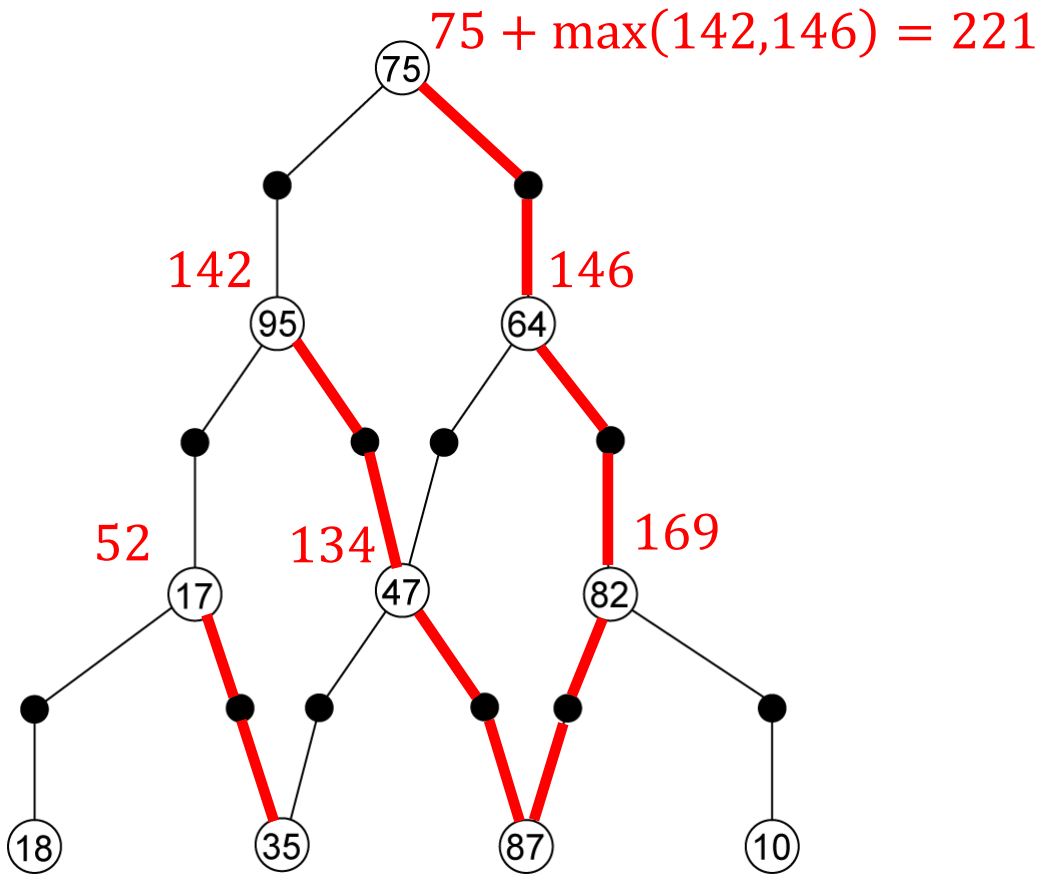
Iteration 1



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

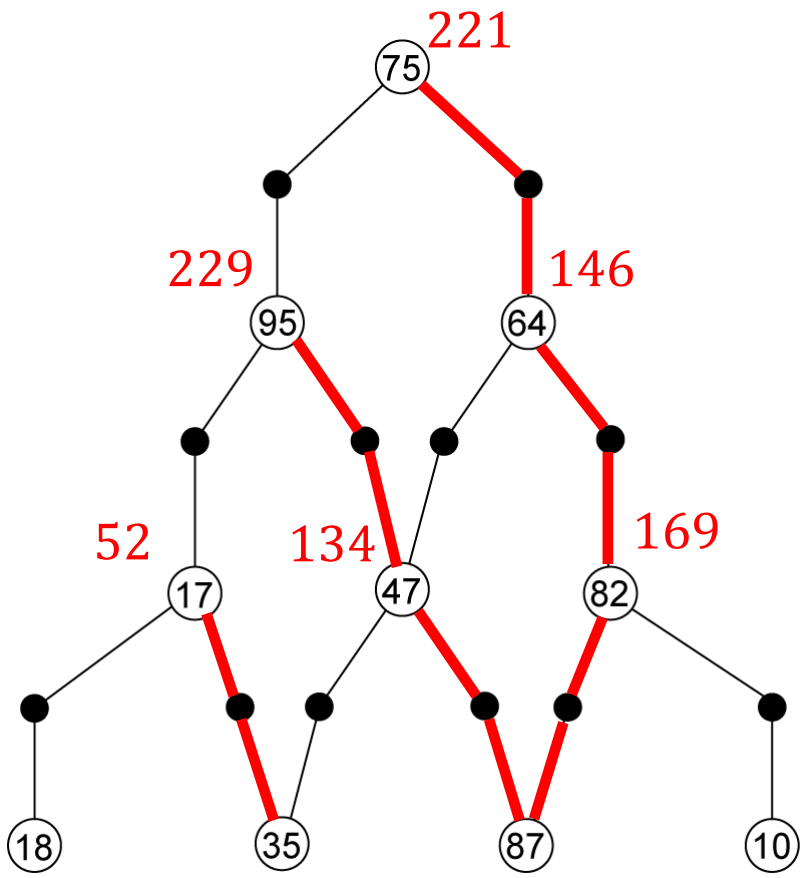
Iteration 2



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

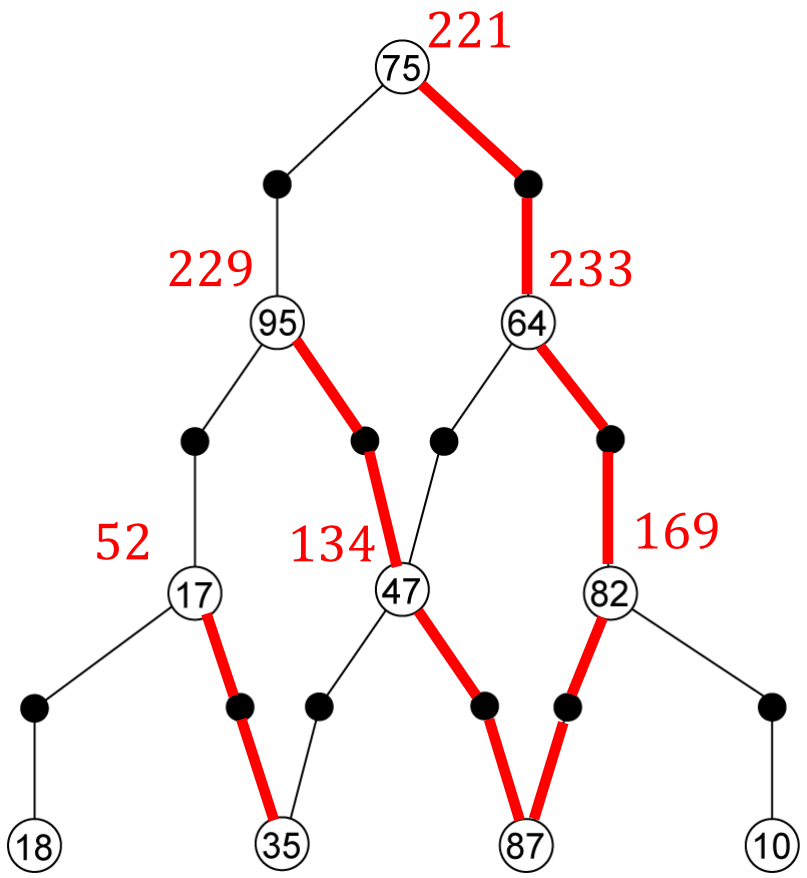
Iteration 2



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

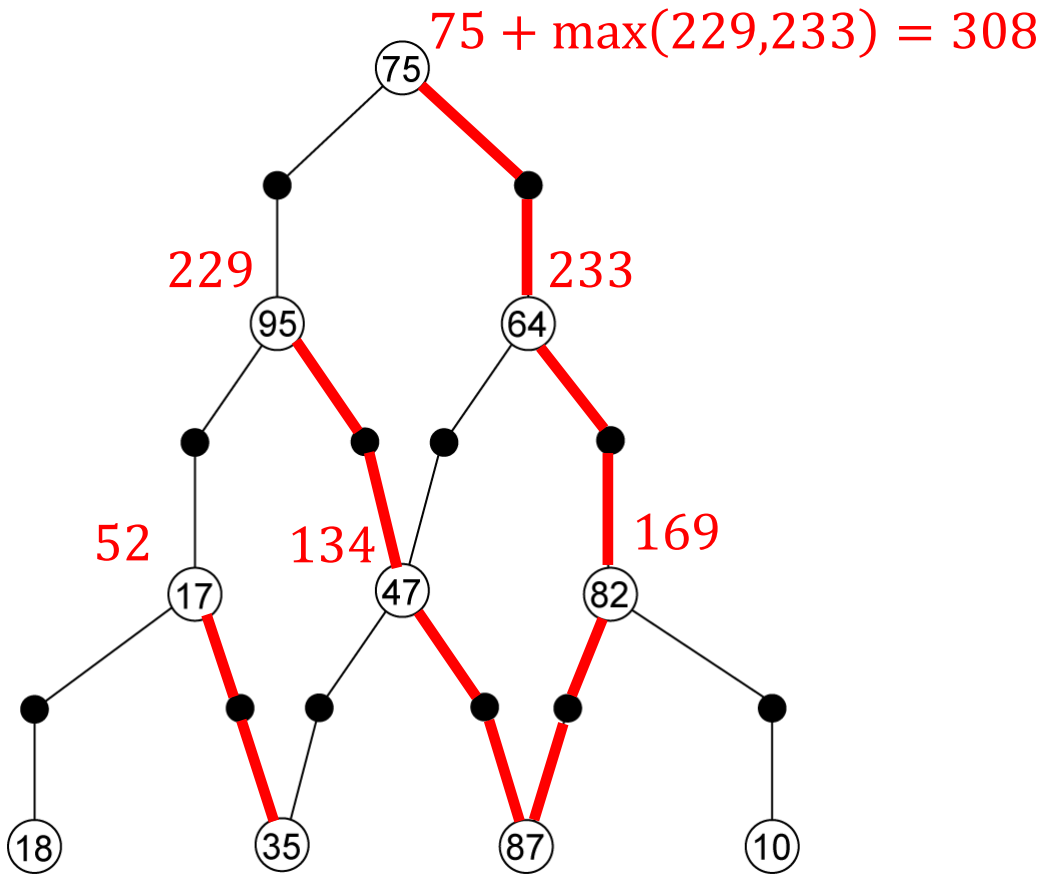
Iteration 2



Value Iteration Recap

Use *value iteration* to find the optimal value function and the optimal policy for the triangle on the right

Iteration 3



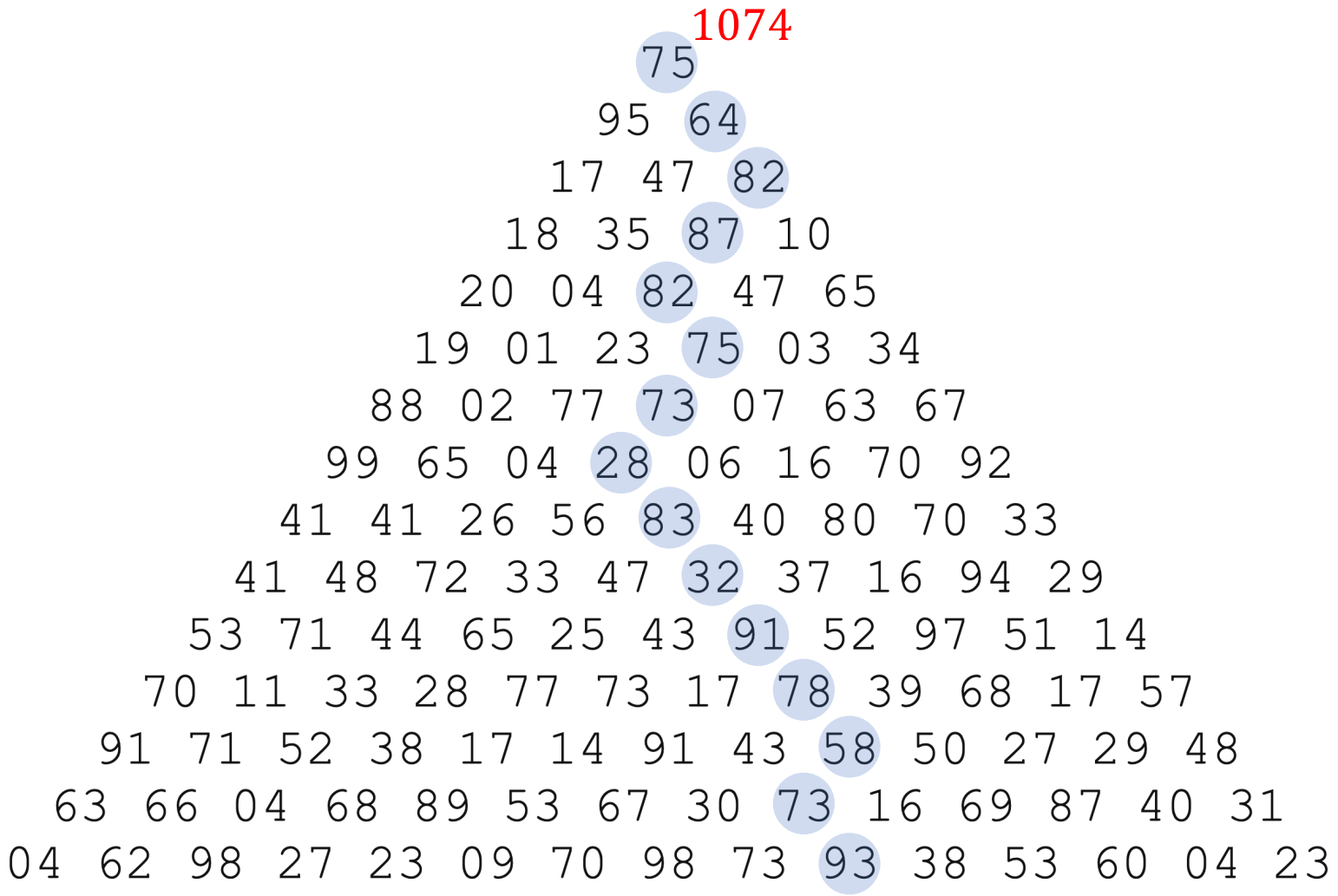
Exercise

Write a program that uses *value iteration* to maximize the total from top to bottom

```
                75
               95 64
              17 47 82
             18 35 87 10
            20 04 82 47 65
           19 01 23 75 03 34
          88 02 77 73 07 63 67
         99 65 04 28 06 16 70 92
        41 41 26 56 83 40 80 70 33
       41 48 72 33 47 32 37 16 94 29
      53 71 44 65 25 43 91 52 97 51 14
     70 11 33 28 77 73 17 78 39 68 17 57
    91 71 52 38 17 14 91 43 58 50 27 29 48
   63 66 04 68 89 53 67 30 73 16 69 87 40 31
  04 62 98 27 23 09 70 98 73 93 38 53 60 04 23
```

Solution

Write a program that uses *value iteration* to maximize the total from top to bottom



Bellman Optimality Equation

Solution Methods

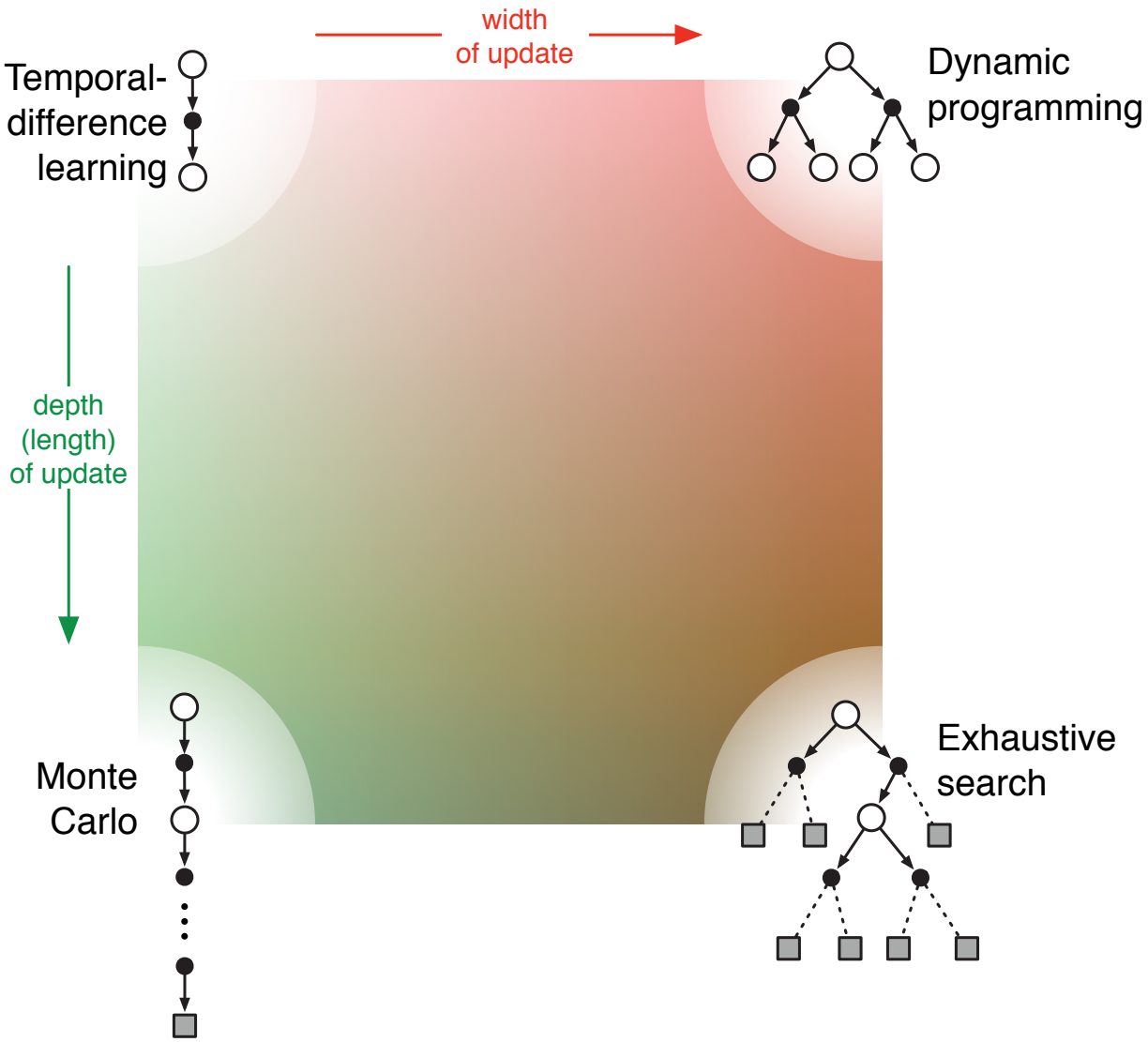
- Policy Iteration
- Value Iteration

Model-based algorithms (*planning*)

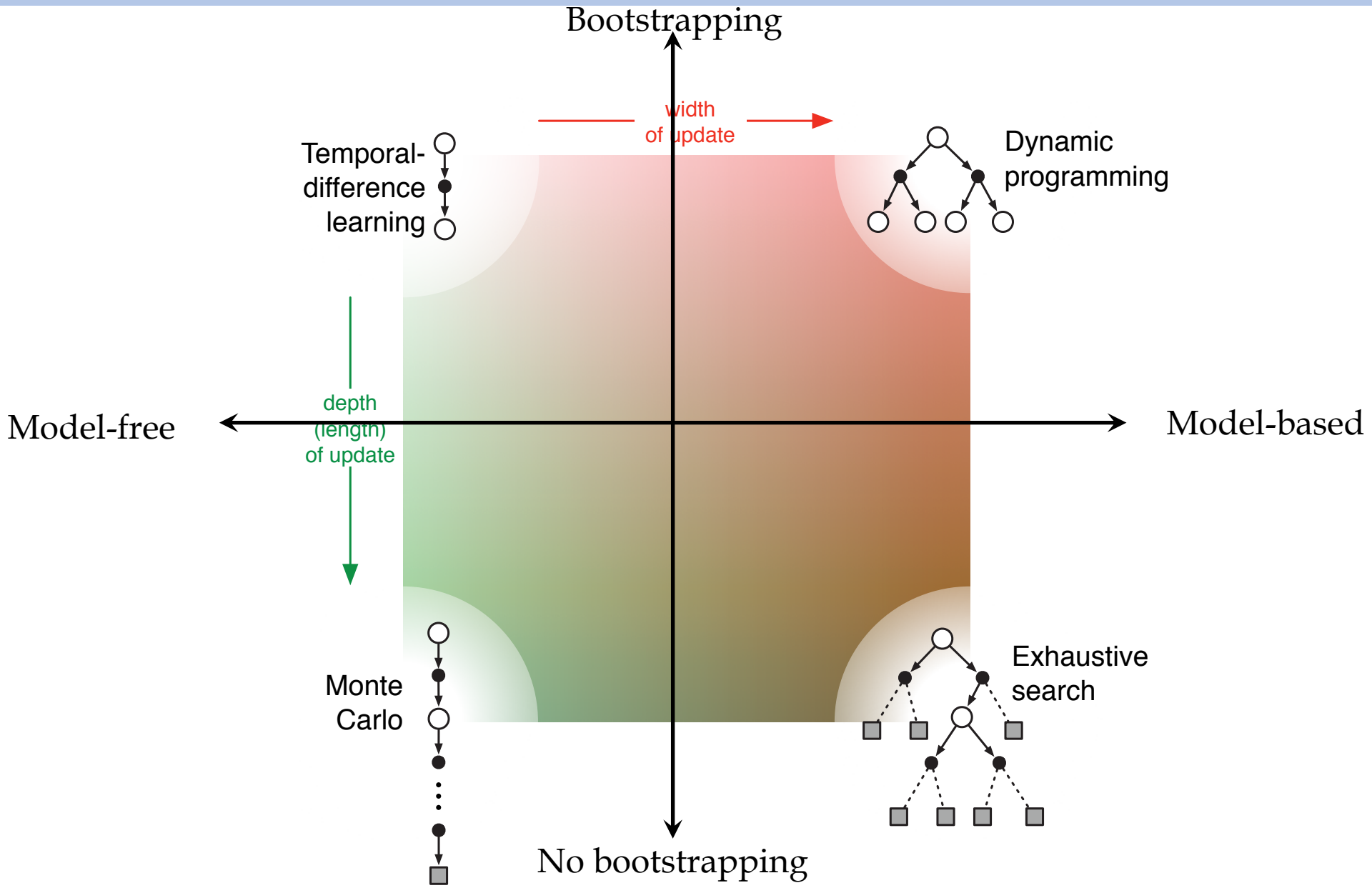
- Monte Carlo
- Temporal Difference

Model-free algorithms (*learning*)

Bellman Optimality Equation



Bellman Optimality Equation



Bellman Optimality Equation

Solution Methods

- Policy Iteration
- Value Iteration
- Monte Carlo
- Temporal Difference

Monte Carlo Methods

Do not need a model of the world, instead learn from *experience* – sample sequences of states, actions, and rewards through interaction

Assume experience is divided into *episodes* that eventually terminate, no matter what actions are selected

Can be used for *prediction* and *control*

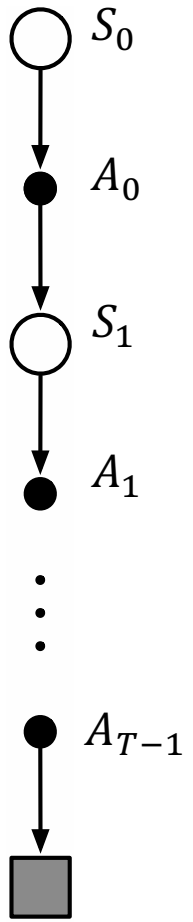
Monte Carlo Prediction

Idea: Given a policy $\pi(a|s)$, learn $v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$ by *sampling* returns

Input: $\pi(s) \in \mathcal{A}(s)$

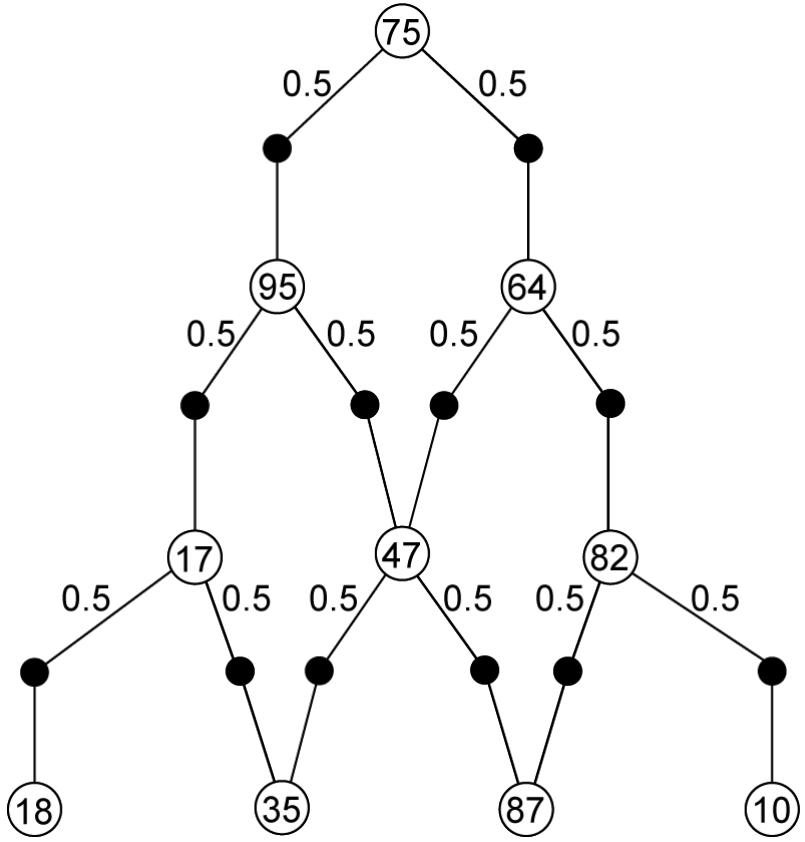
Random initialization:
$V(s) \in \mathbb{R}, \forall s \in \mathcal{S}$
Returns$(s) \leftarrow$ an empty list, $\forall s \in \mathcal{S}$

Loop for each episode:
Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode:
$G \leftarrow \gamma G + R_{t+1}$
Append G to Returns(S_t)
V$(S_t) \leftarrow$ average(Returns(S_t))



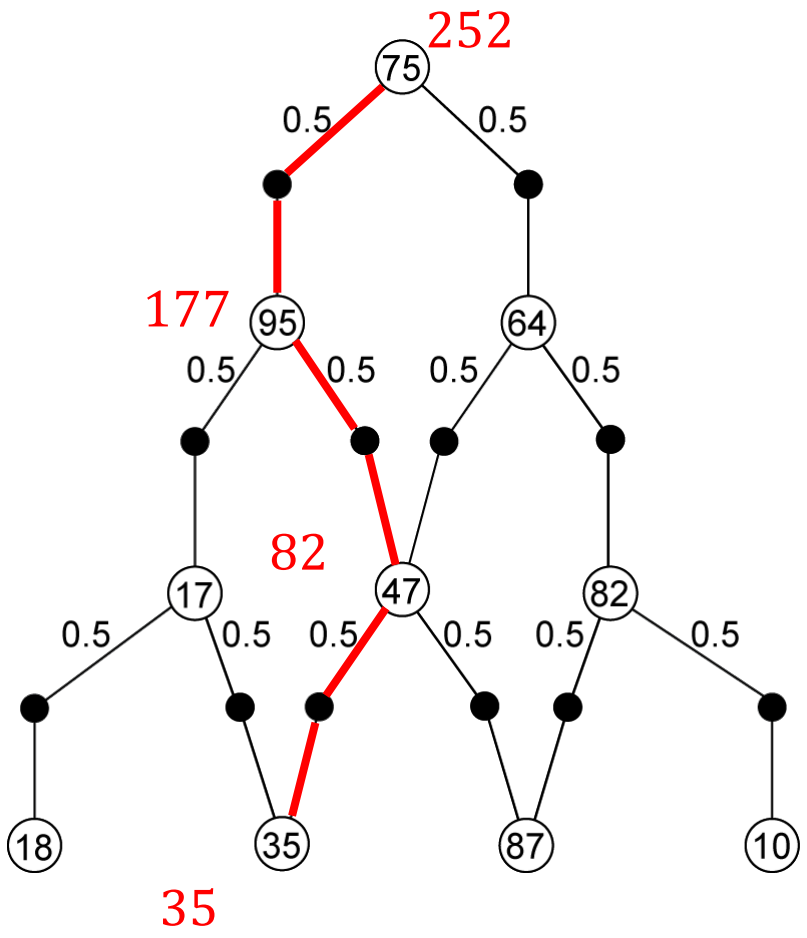
Monte Carlo Prediction

Use Monte Carlo method to evaluate state values of the triangle under a random walk policy



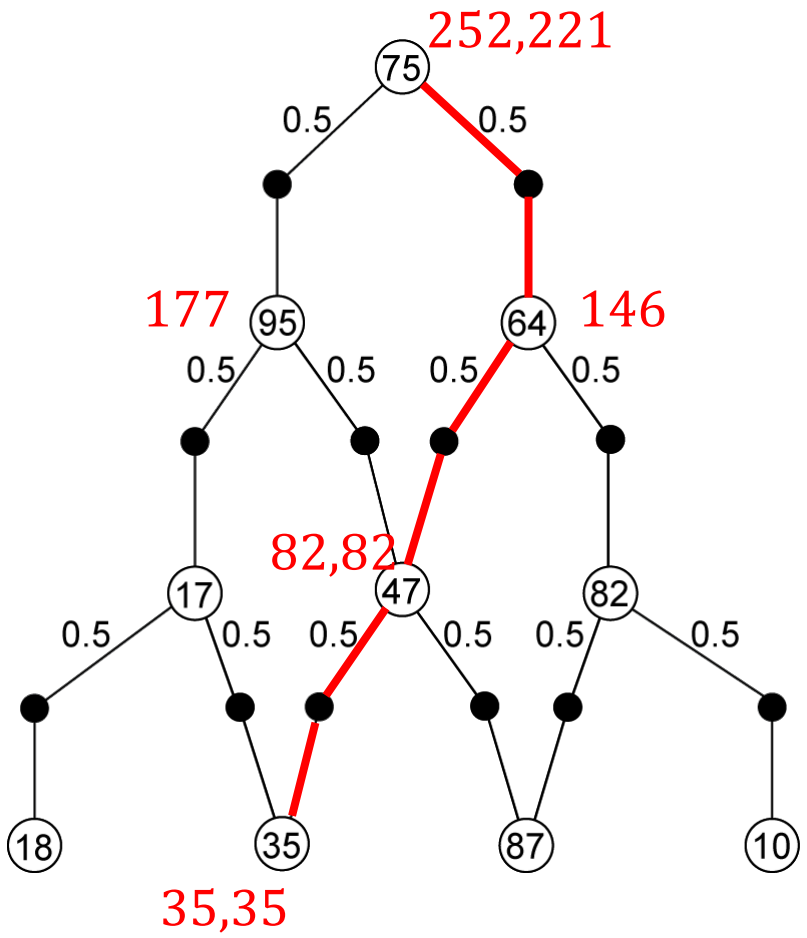
Monte Carlo Prediction

Episode 1



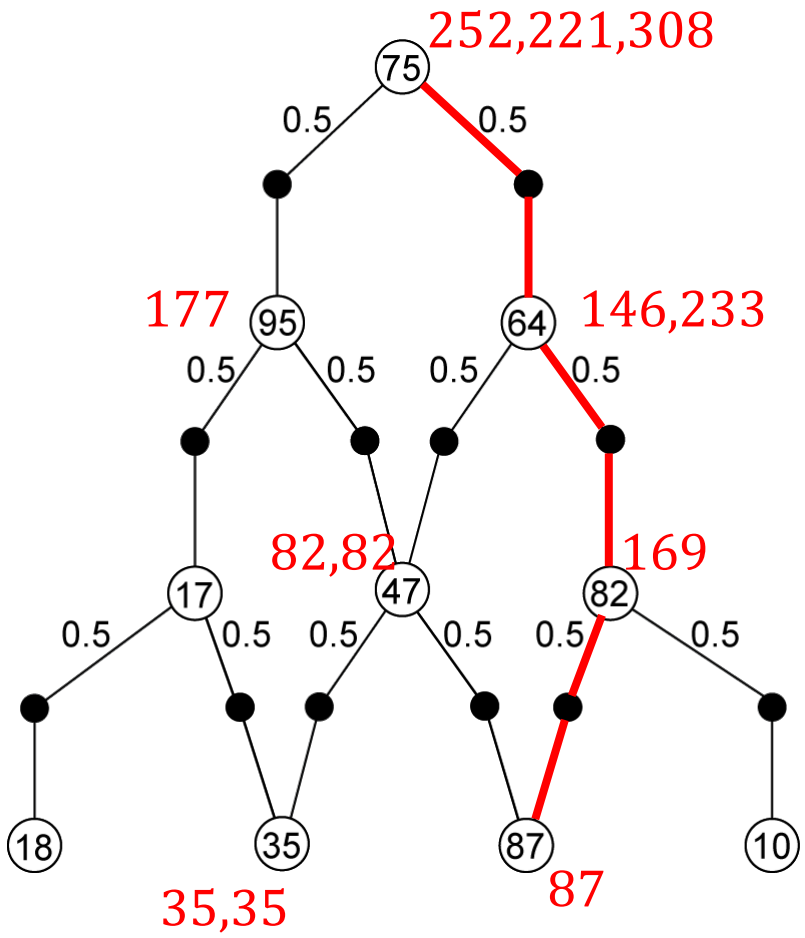
Monte Carlo Prediction

Episode 2



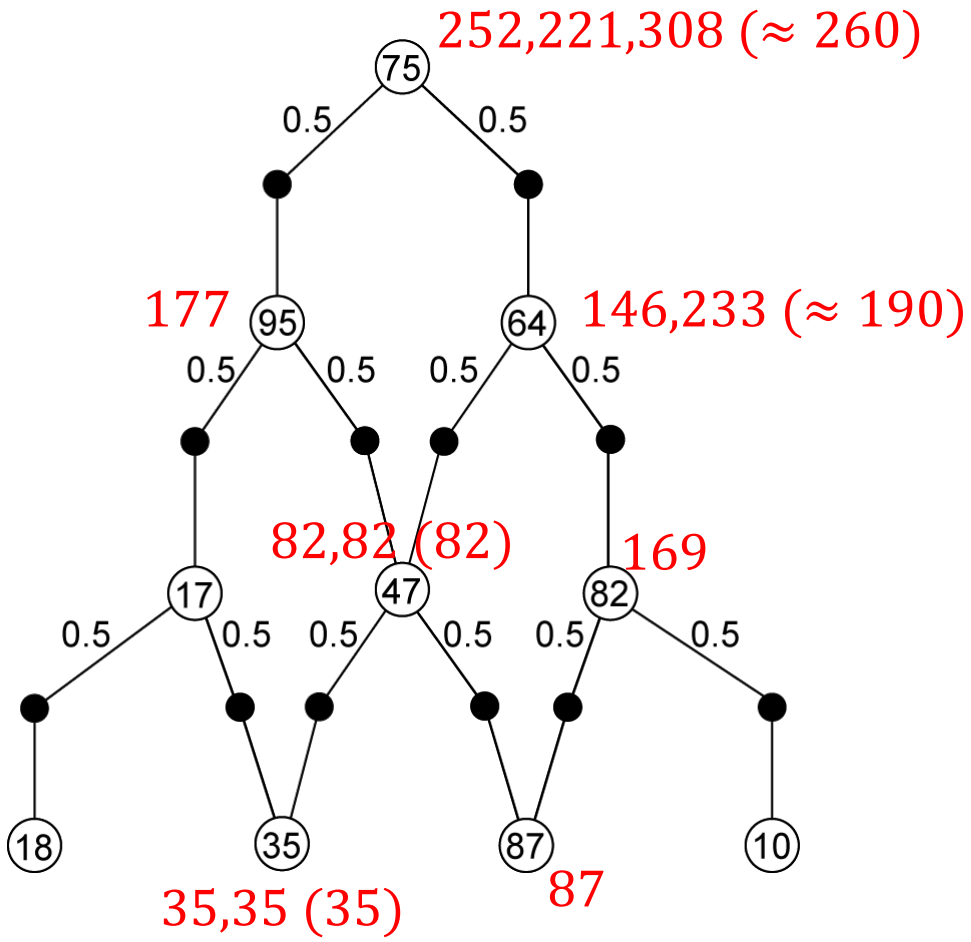
Monte Carlo Prediction

Episode 3



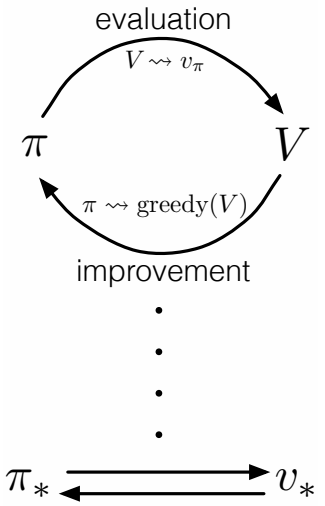
Monte Carlo Prediction

Average!



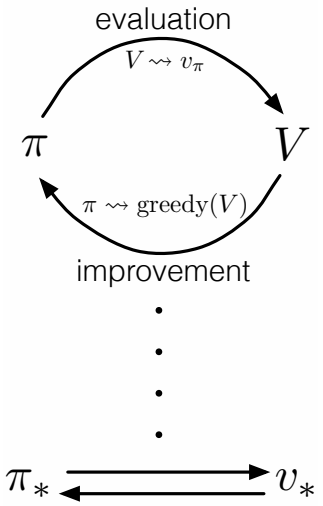
Monte Carlo Control

Idea: To find the optimal policy, proceed using the same approach as Dynamic Programming



Monte Carlo Control

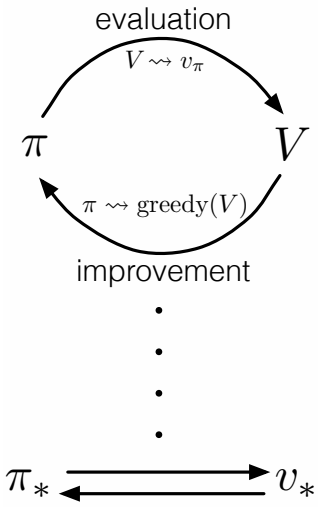
Idea: To find the optimal policy, proceed using the same approach as Dynamic Programming



But: To compute greedy policy from *state values*, we must look ahead one step to choose best action. This requires knowing the model of the world!

Monte Carlo Control

Idea: To find the optimal policy, proceed using the same approach as Dynamic Programming

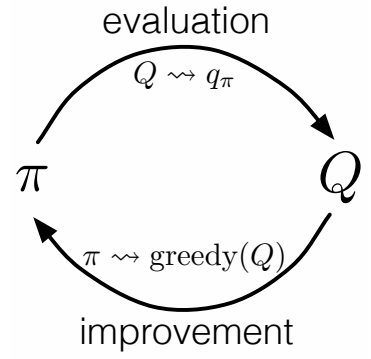


But: To compute greedy policy from *state values*, we must look ahead one step to choose best action. This requires knowing the model of the world!

Solution: Use *action values*, $q_\pi(s,a)$ instead of *state values*, $v_\pi(s)$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \qquad \textbf{Evaluation}$$

$$\pi(s) \doteq \underset{a}{\text{argmax}}\ q(s,a) \qquad \textbf{Improvement}$$



Monte Carlo Control

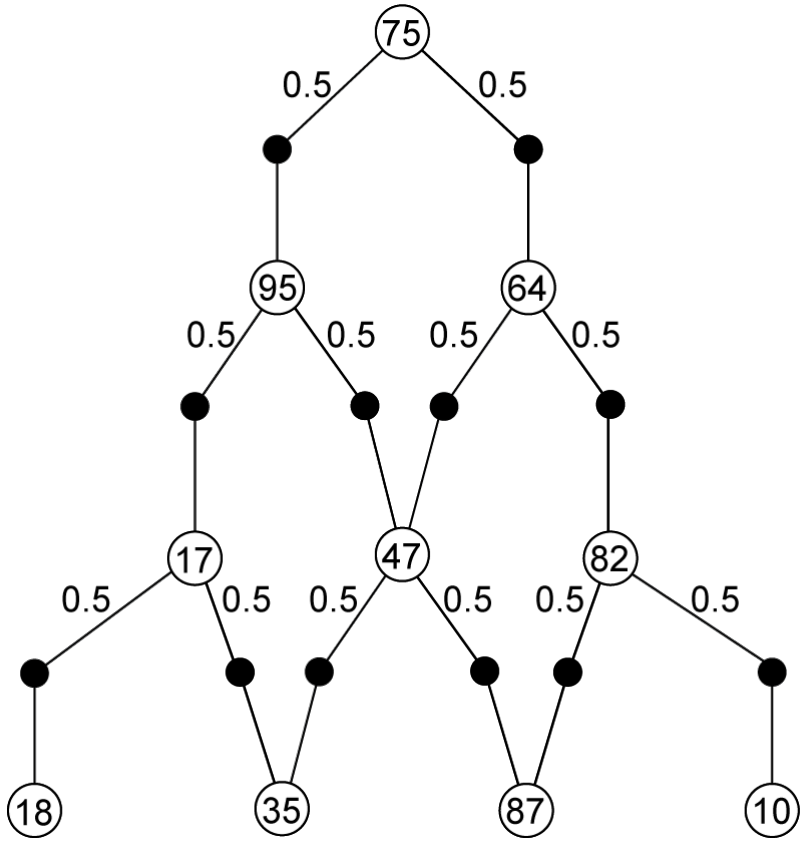
Input: $\pi(s) \in \mathcal{A}(s)$

Random initialization:
$Q(s,a) \in \mathbb{R}, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
Returns$(s,a) \leftarrow$ an empty list, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop for each episode:
Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode:
$G \leftarrow \gamma G + R_{t+1}$
Append G to Returns(S_t, A_t)
$Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
$\pi(S_t) \leftarrow \underset{a}{\text{argmax}}\ Q(S_t, a)$

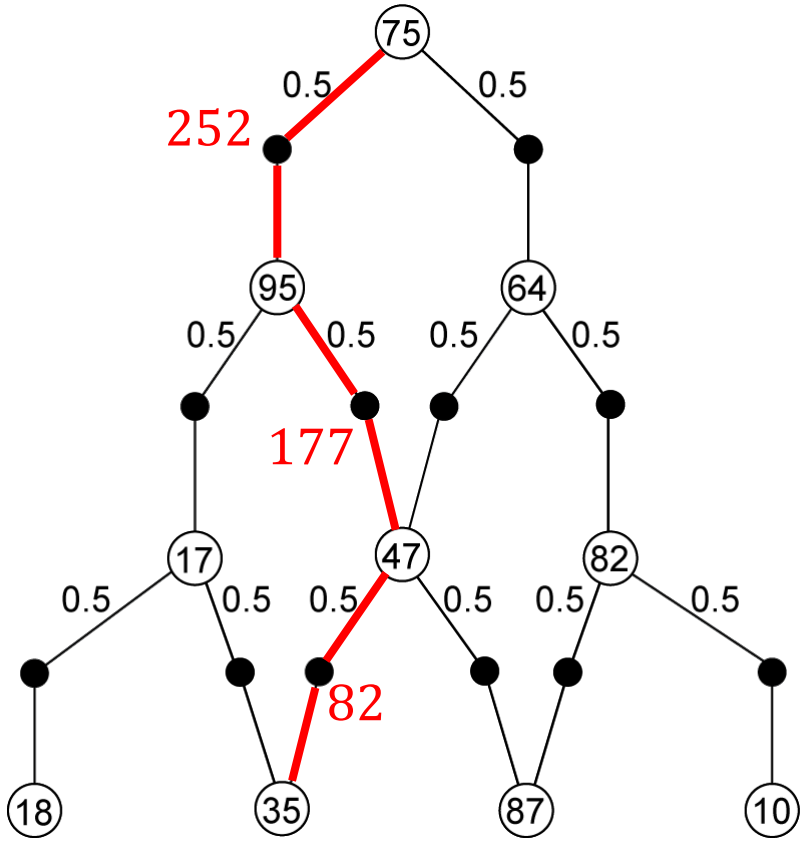
Group discussion

Use Monte Carlo method to evaluate action values for the triangle under a random walk policy



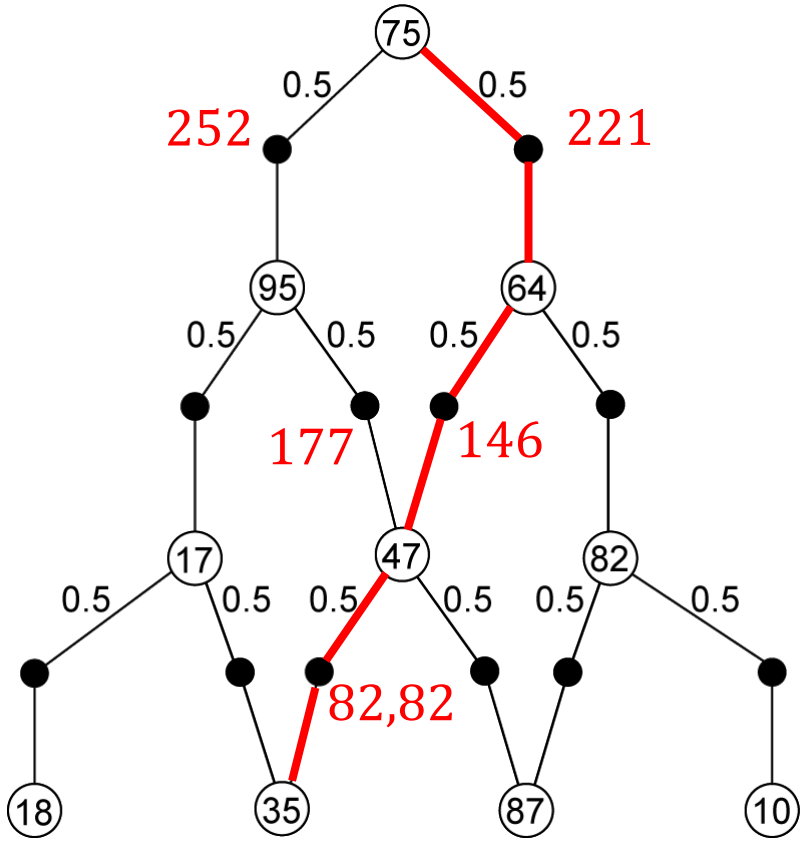
Group discussion

Episode 1



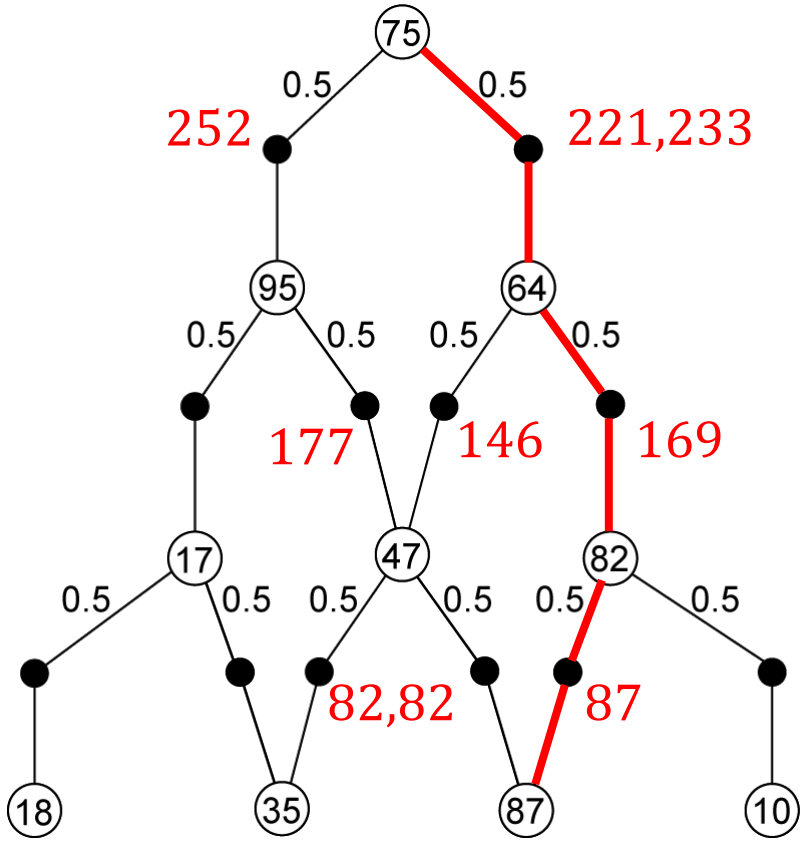
Group discussion

Episode 2



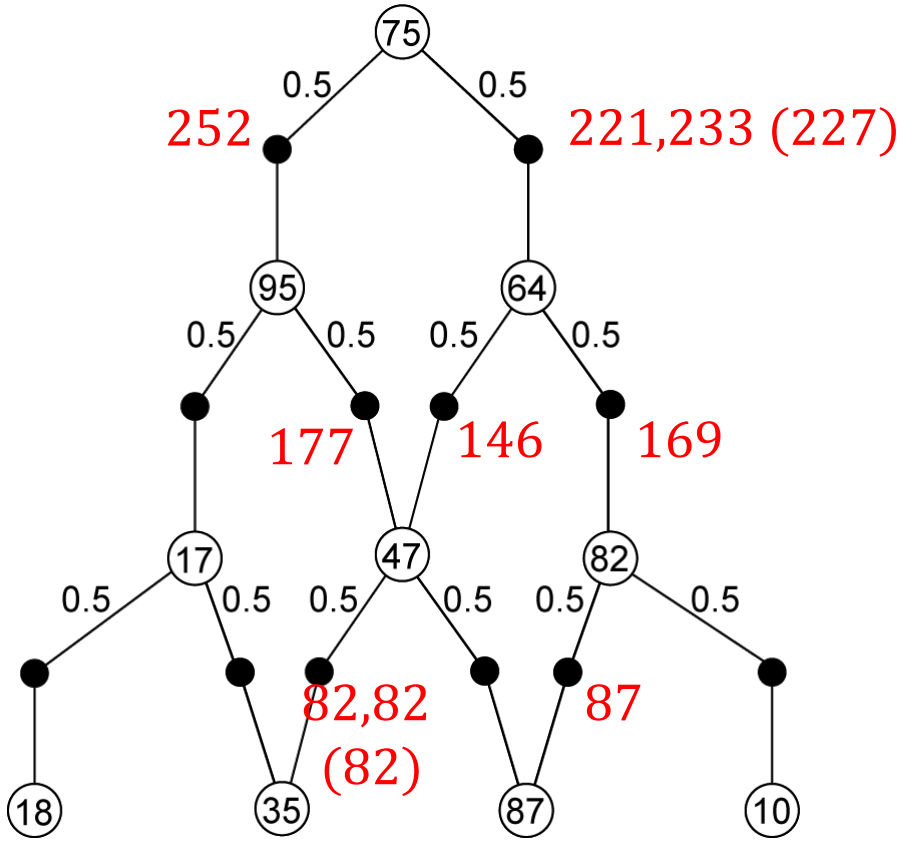
Group discussion

Episode 3



Group discussion

Average!



Monte Carlo Control

Input: $\pi(s) \in \mathcal{A}(s)$

Random initialization:
$Q(s,a) \in \mathbb{R}, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
Returns$(s,a) \leftarrow$ an empty list, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop for each episode:
Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode:
$G \leftarrow \gamma G + R_{t+1}$
Append G to Returns(S_t, A_t)
$Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
$\pi(S_t) \leftarrow \underset{a}{\text{argmax}}\ Q(S_t, a)$

Convergence requires coverage!

Monte Carlo Control

Convergence requires coverage!

Cannot use greedy policy for sampling episodes but there are two solutions:

- ε-greedy sampling (*on-policy*)
- Importance sampling (*off-policy*)

Off-policy **Monte Carlo Control via** *importance sampling*

Idea:
Use two separate policies – one being learning is called the *target policy,* $\pi(a|s)$ and one used for sampling is called the *behavior policy,* $b(a|s)$

How to transform $v_b(s,a)$ to $v_\pi(s,a)$?

$$v_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_b[\rho_{t:T-1}G_t|S_t = s]$$

where $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$ **Importance sampling ratio**

Bellman Optimality Equation

Solution Methods

- Policy Iteration
- Value Iteration
- Monte Carlo
- Temporal Difference

Temporal Difference

Do not need a model of the world, instead learn from *experience* (like Monte Carlo)

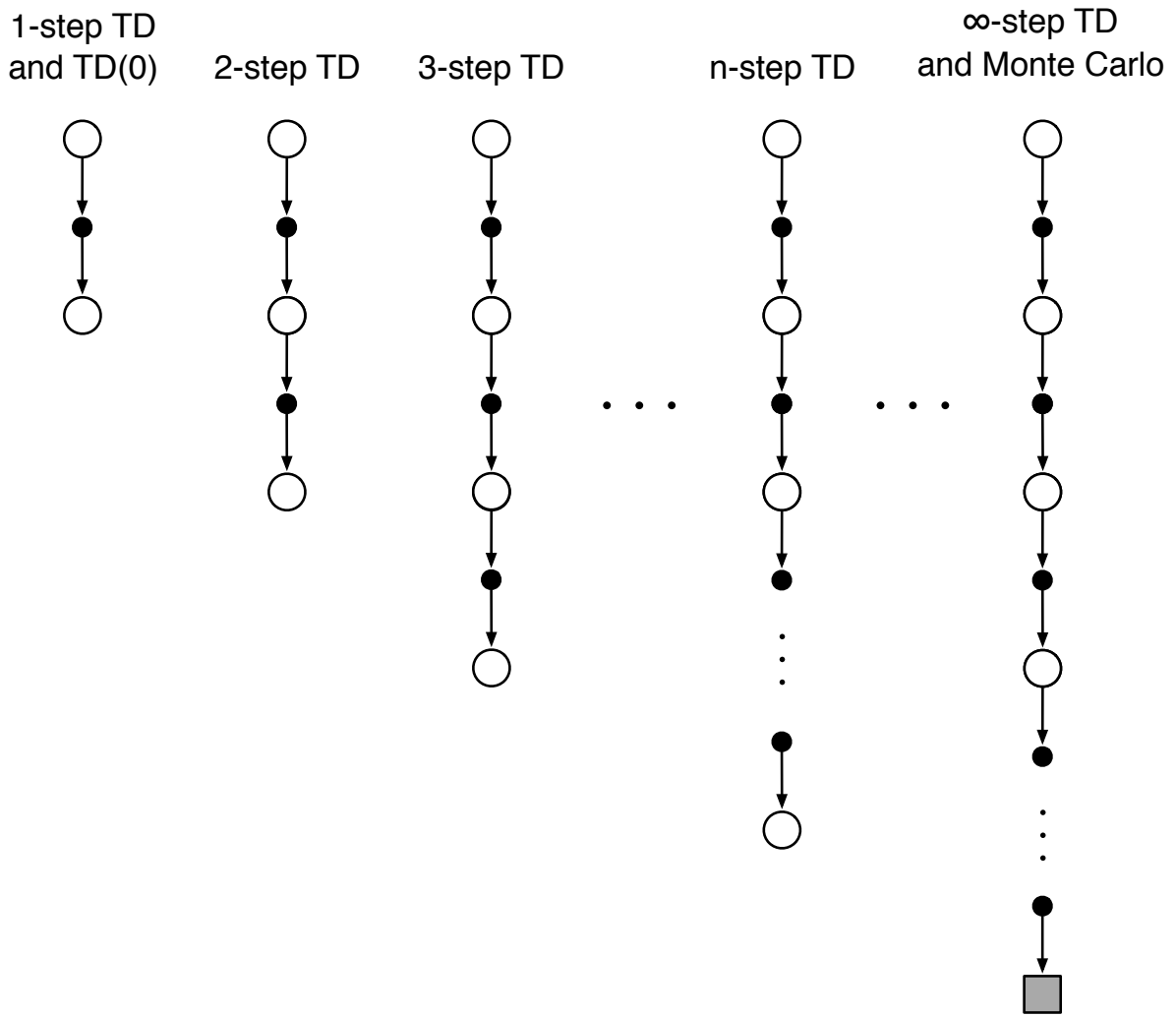
Update value estimates based on other learned estimates (like Dynamic programming)

Can be used for *prediction* and *control*

Temporal Difference

Need to bootstrap

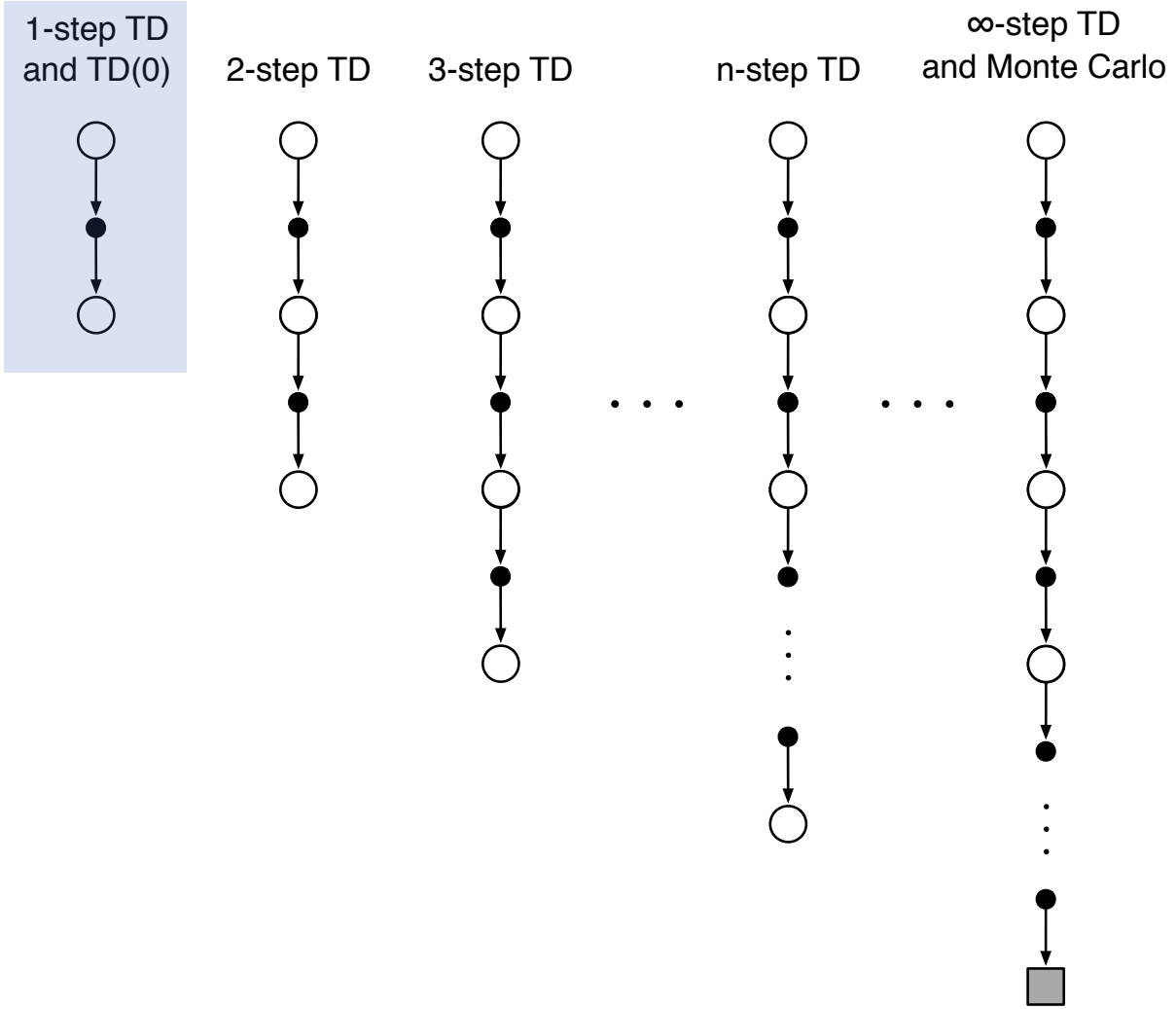
No need to bootstrap



Temporal Difference

Need to bootstrap

No need to bootstrap



Temporal Difference Prediction

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) + V(S_t)]$$

Temporal Difference Prediction

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Reward prediction error, δ_t

Temporal Difference Prediction

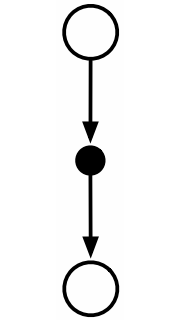
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Input: $\pi(s) \in \mathcal{A}(s)$

Random initialization:
$V(s) \in \mathbb{R}, \forall s \in \mathcal{S}$

Loop for each episode:
Initialize S
Loop for each step of episode:
Take action A from $\pi(S)$
Observe S', R
$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
$S \leftarrow S'$



TD(0)

Temporal Difference Control

Like Monte Carlo Control, there are *on-policy* and *off-policy* methods for TD control:

- SARSA (*on-policy*)
- Q-learning (*off-policy*)

On-policy **TD Control via** *SARSA*

Random initialization:
$Q(s,a) \in \mathbb{R}, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

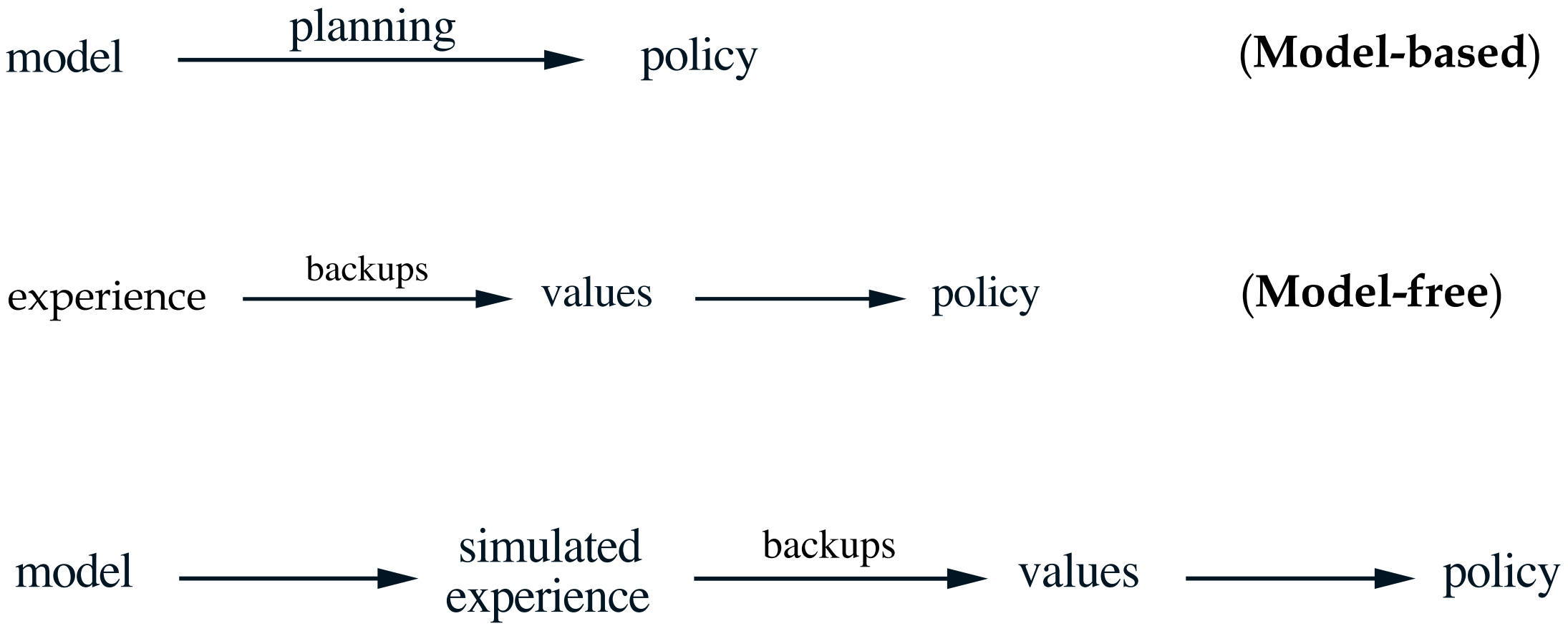
Loop for each episode:
    Initialize S
    Choose action A using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, Observe S', R
        Choose action A' using policy derived from Q (e.g., ε-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'; A \leftarrow A'$

Off-policy **TD Control via** *Q-learning*

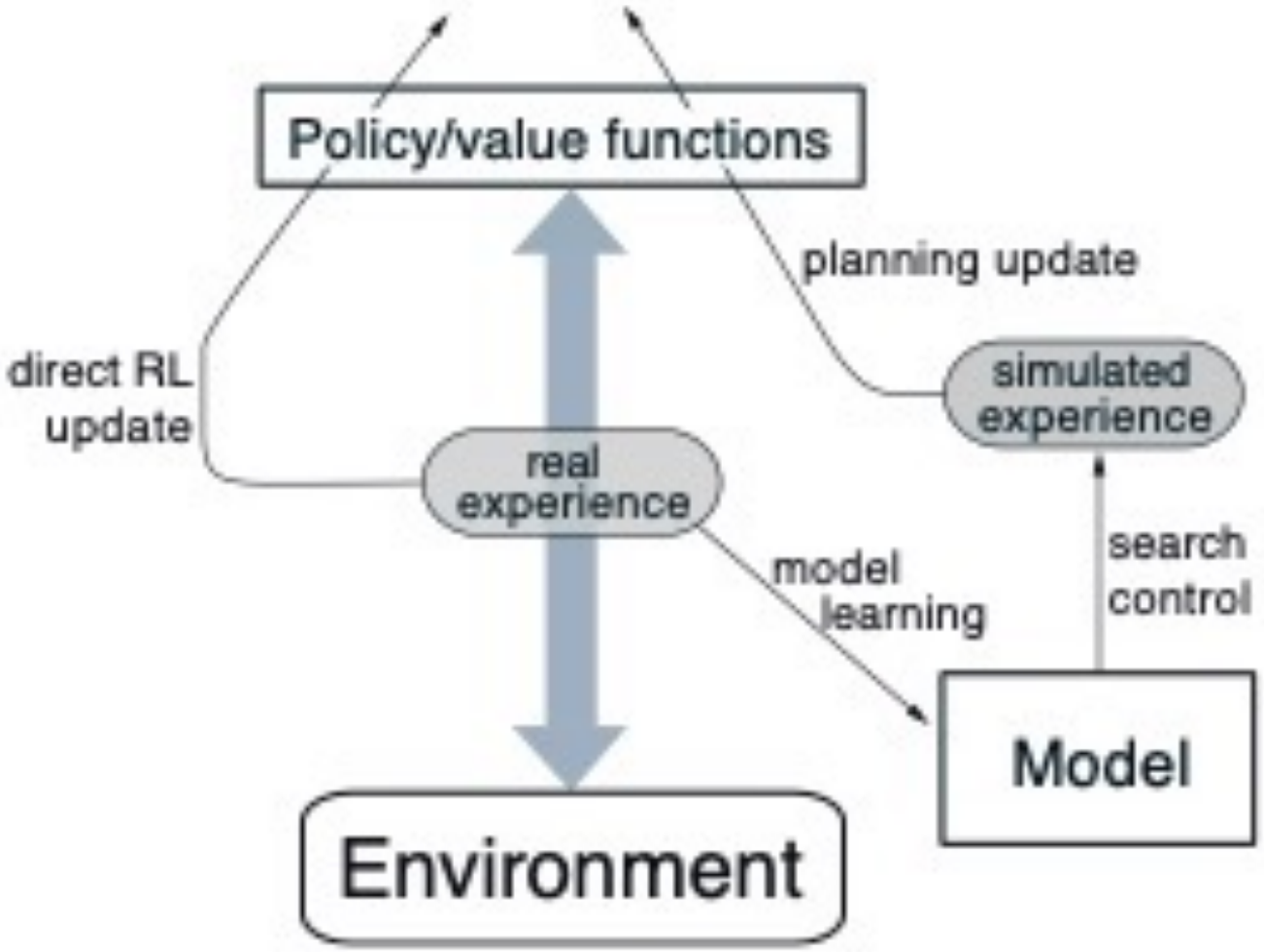
Random initialization:
$Q(s,a) \in \mathbb{R}, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$

Loop for each episode:
Initialize S
Loop for each step of episode:
Choose action A using policy derived from Q (e.g., ε-greedy)
Take action A, Observe S', R
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
$S \leftarrow S'; A \leftarrow A'$

Unified view of model-based and model-free methods



Dyna: Integrated planning and learning



The end