ELTE | FACULTY OF INFORMATICS

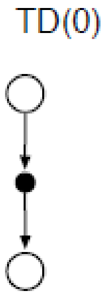
N-STEP BOOTSTRAPPING

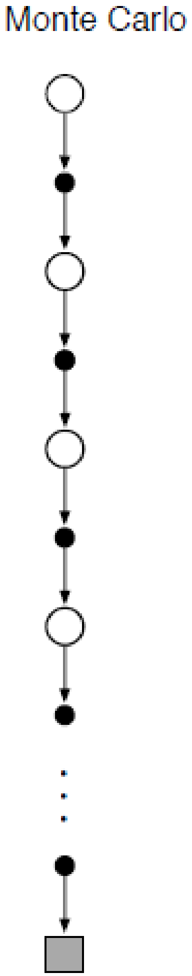
Deep Reinforcement Learning

Balázs Nagy, PhD

ELTE | IK
DEPARTMENT OF ARTIFICIAL INTELLIGENCE

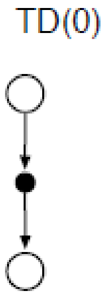
N-step Bootstrapping

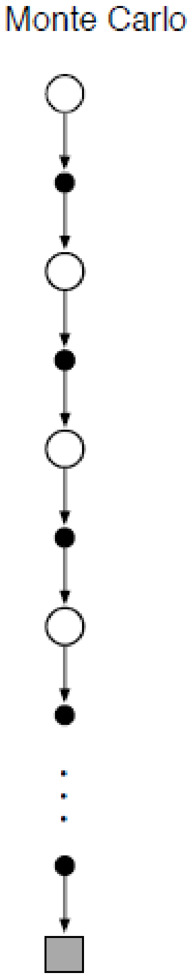




N-step Bootstrapping

- Fast update
- Can be done on the run

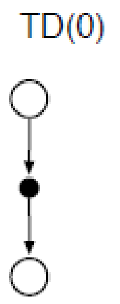




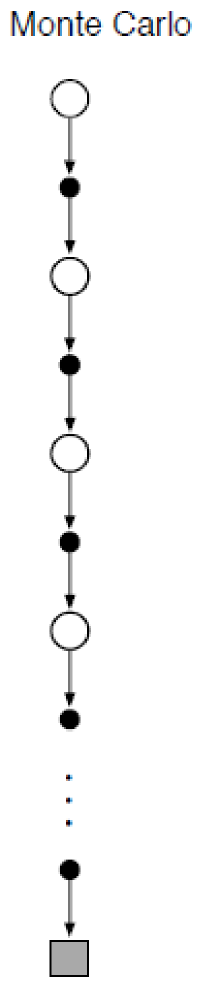
- Update only at the end of episode
- Better approximate over time

N-step Bootstrapping

- Fast update
- Can be done on the run



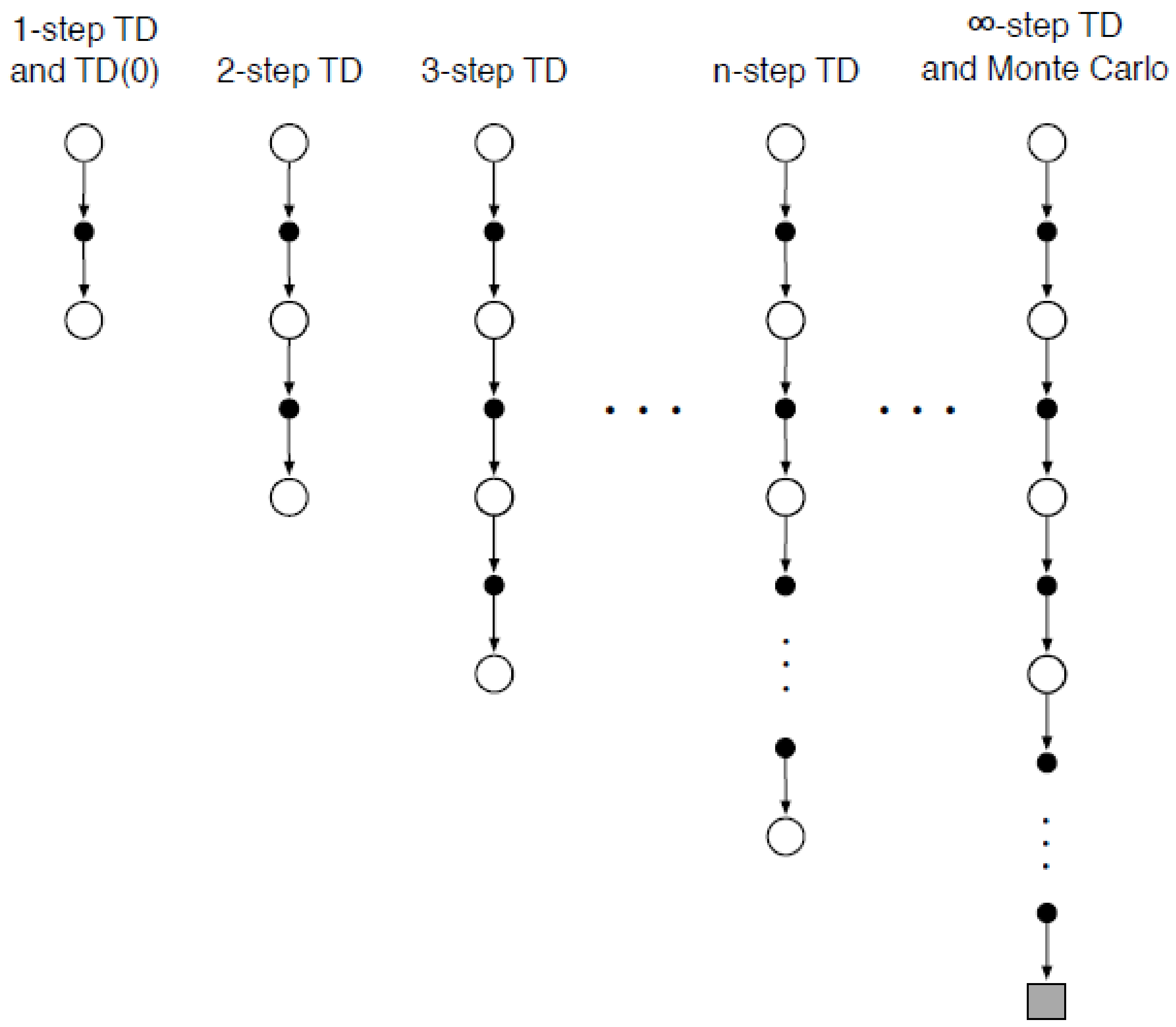
?



- Update only at the end of episode
- Better approximate over time

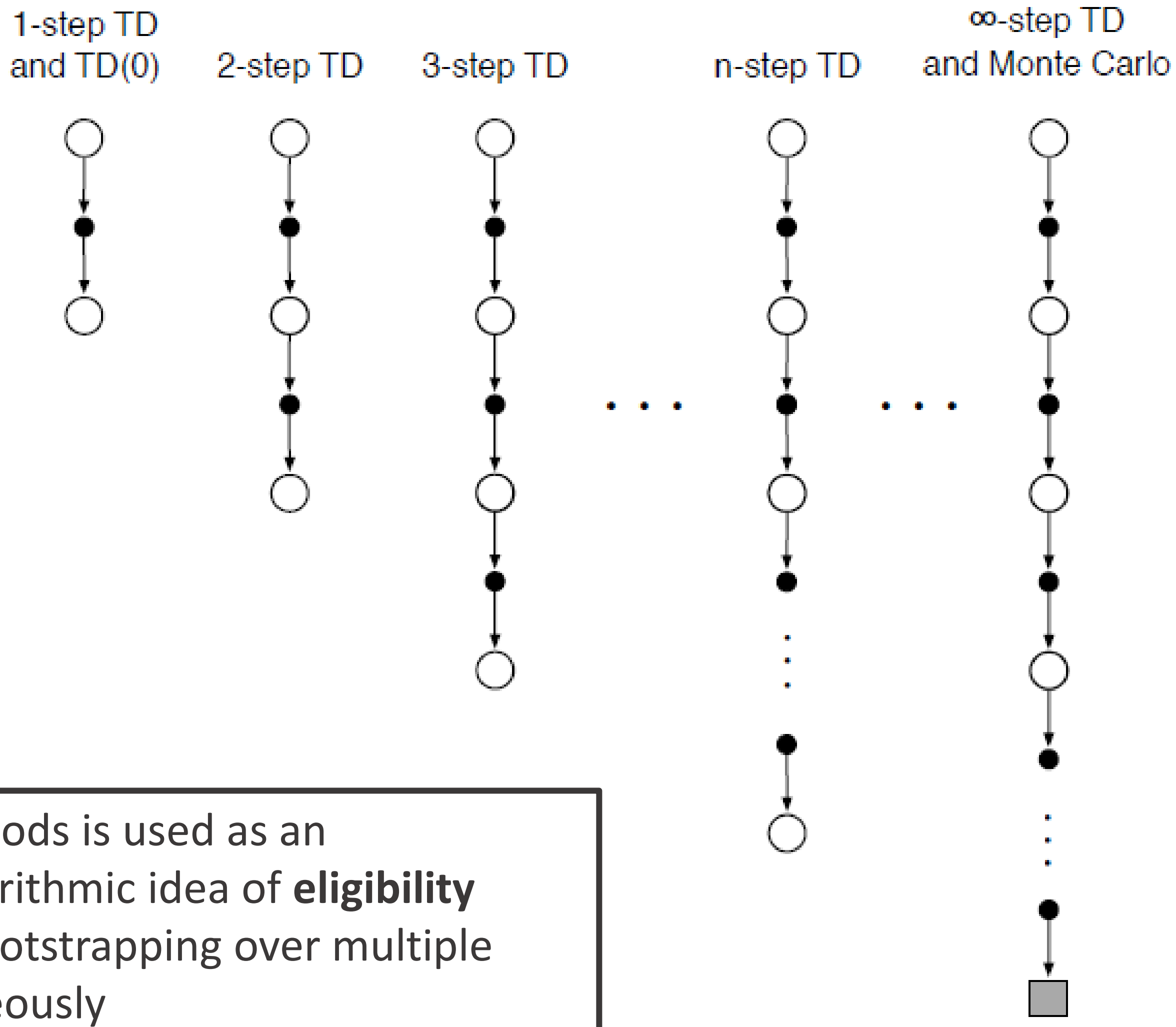
N-step Bootstrapping

- Fast update
- Can be done on the run



- Update only at the end of episode
- Better approximate over time

N-step Bootstrapping



- Fast update
- Can be done on the run

- Update only at the end of episode
- Better approximate over time

The idea of n-step methods is used as an introduction to the algorithmic idea of **eligibility traces**, which enable bootstrapping over multiple time intervals simultaneously

Complete Return

- Monte Carlo

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Complete Return

- One-step return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- Monte Carlo

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Complete Return

- One-step return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- Two-step return

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- Monte Carlo

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Complete Return

- One-step return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- Two-step return

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- n-step return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- Monte Carlo

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

Complete Return

- One-step return

$$G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$$

- Two-step return

$$G_{t:t+2} \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

- n-step return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

$$\boxed{(G_{t:t+n} \doteq G_t \text{ if } t+n \geq T)}$$

- Monte Carlo

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$$

State value function for n-step TD prediction

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\left[G_{t:t+n} - V_{t+n-1}(S_t)\right], \qquad 0 \leq t < T$$

- n-step returns for $n > 1$ involve future rewards and states that are not available at the time of transition from t to $t + 1$
- No real algorithm can use the n-step return until after it has seen R_{t+n} and computed V_{t+n-1}
- The values of all other states remain unchanged: $V_{t+n}(s) = V_{t+n-1}(s)$, for all $s \neq S_t$

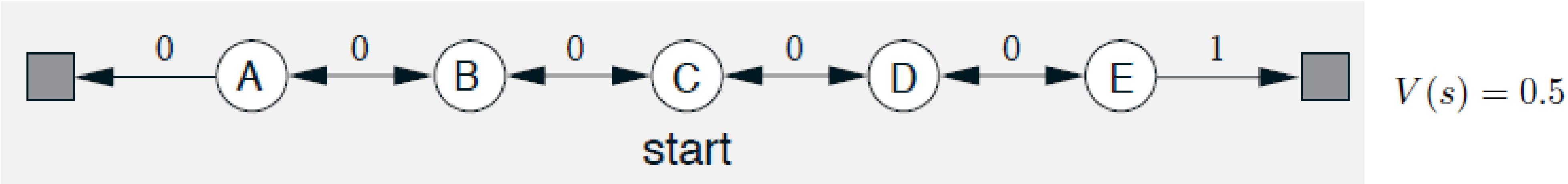
Pseudocode

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

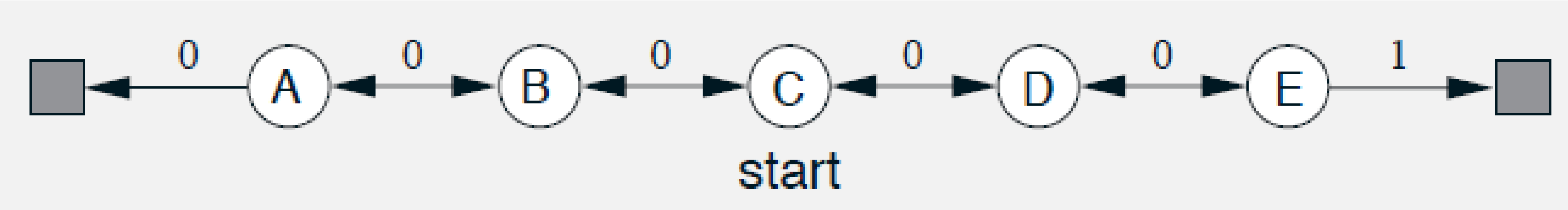
Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    Loop for $t = 0, 1, 2, \dots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then $T \leftarrow t+1$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
    Until $\tau = T - 1$

n-step TD Methods on the Random Walk



- The first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1

n-step TD Methods on the Random Walk

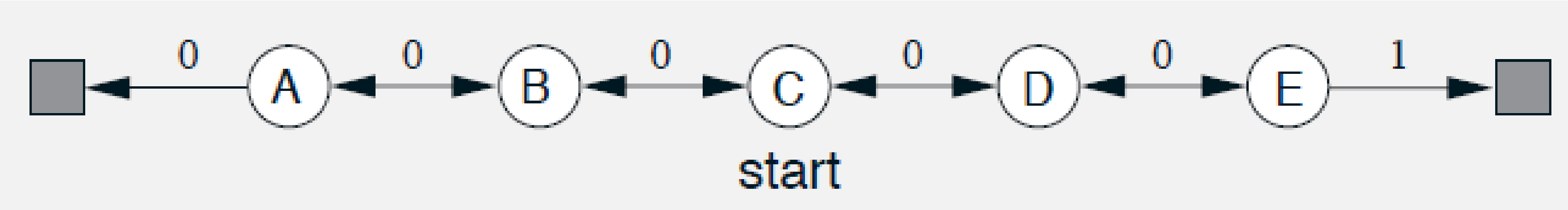


$V(s) = 0.5$

- The first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1
- Which values would change?

Algorithm	Changed Values
1-step TD	
2-step TD	
3-step TD	

n-step TD Methods on the Random Walk



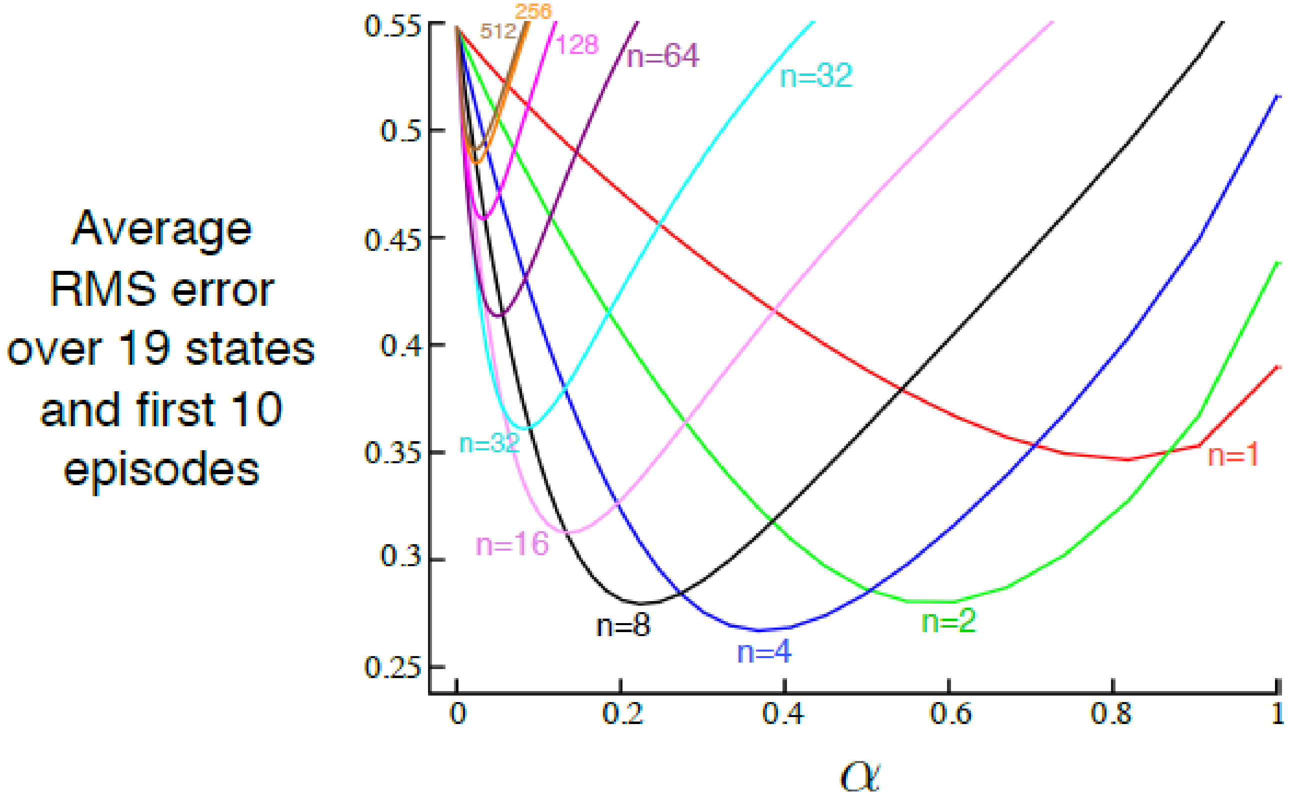
$V(s) = 0.5$

- The first episode progressed directly from the center state, C, to the right, through D and E, and then terminated on the right with a return of 1
- Which values would change?

Algorithm	Changed Values
1-step TD	V(E)
2-step TD	V(E), V(D)
3-step TD	V(E), V(D), V(C)

Empirical test for a larger random walk process

The **generalization** of TD and Monte Carlo methods to n-step methods can potentially **perform better** than either of the two extreme methods



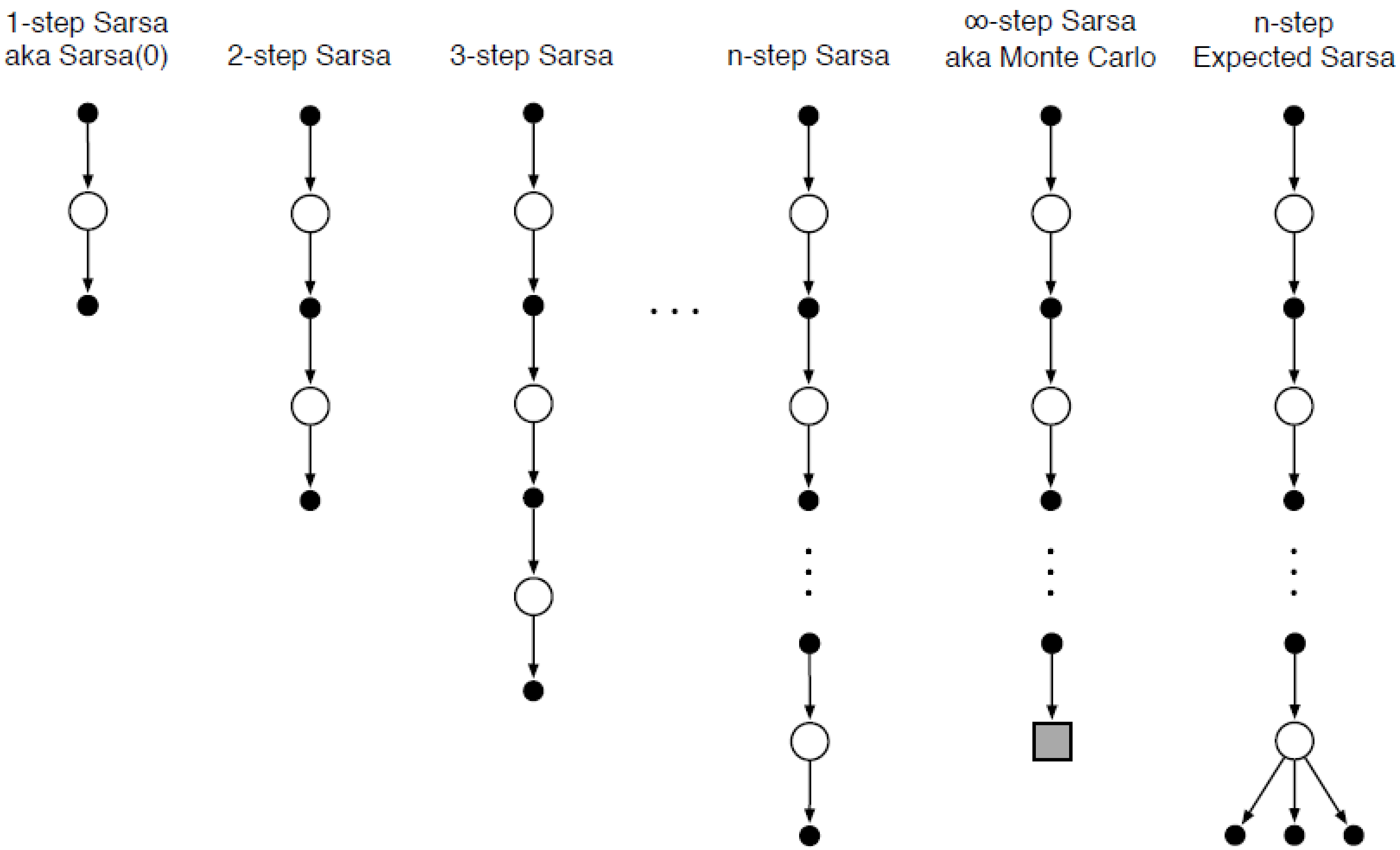
Pseudocode

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

```
Loop for each episode:
    Initialize and store S_0 ≠ terminal
    Select and store an action A_0 ~ π(·|S_0)
    T ← ∞
    Loop for t = 0, 1, 2, ... :
    |   If t < T, then:
    |       Take action A_t
    |       Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    |       If S_{t+1} is terminal, then:
    |           T ← t + 1
    |       else:
    |           Select and store an action A_{t+1} ~ π(·|S_{t+1})
    |   τ ← t − n + 1    (τ is the time whose estimate is being updated)
    |   If τ ≥ 0:
    |       G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i
    |       If τ + n < T, then G ← G + γ^n Q(S_{τ+n}, A_{τ+n})                (G_{τ:τ+n})
    |       Q(S_τ, A_τ) ← Q(S_τ, A_τ) + α [G − Q(S_τ, A_τ)]
    |       If π is being learned, then ensure that π(·|S_τ) is ε-greedy wrt Q
    Until τ = T − 1
```

n-step Sarsa



1-step Sarsa
aka Sarsa(0)
2-step Sarsa
3-step Sarsa
n-step Sarsa
∞-step Sarsa
aka Monte Carlo
n-step
Expected Sarsa

n-step Sarsa

- n-step return

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n,$$

- Update function

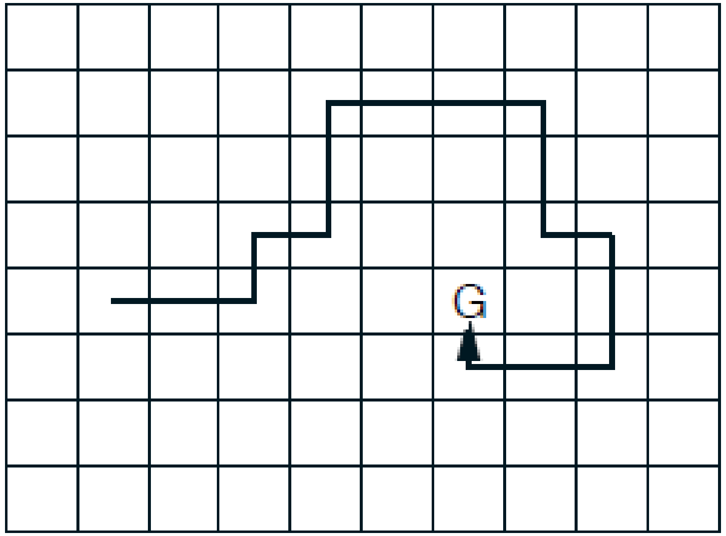
$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(S_t, A_t) \right]$$

$$G_{t:t+n} \doteq G_t \text{ if } t+n \geq T.$$

$$Q_{t+n}(s, a) = Q_{t+n-1}(s, a), \text{for all } s, a \text{ such that } s \neq S_t \text{ or } a \neq A_t$$

Example for speed up learning

Path taken

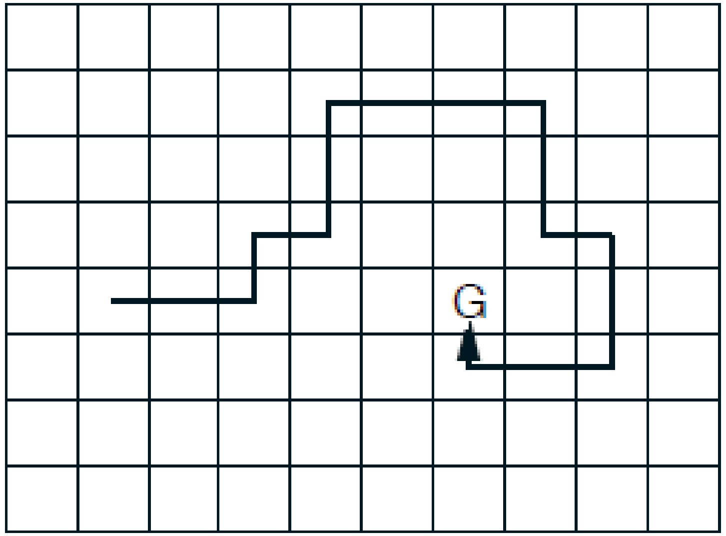


Action values increased by one-step Sarsa

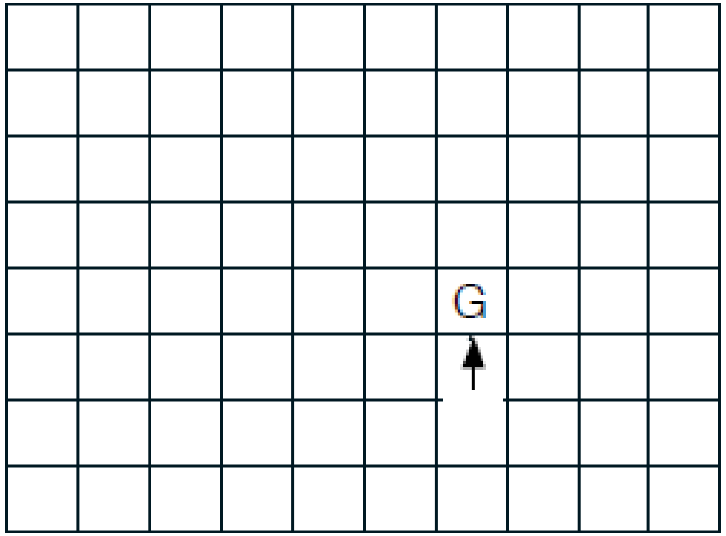
Action values increased by 10-step Sarsa

Example for speed up learning

Path taken

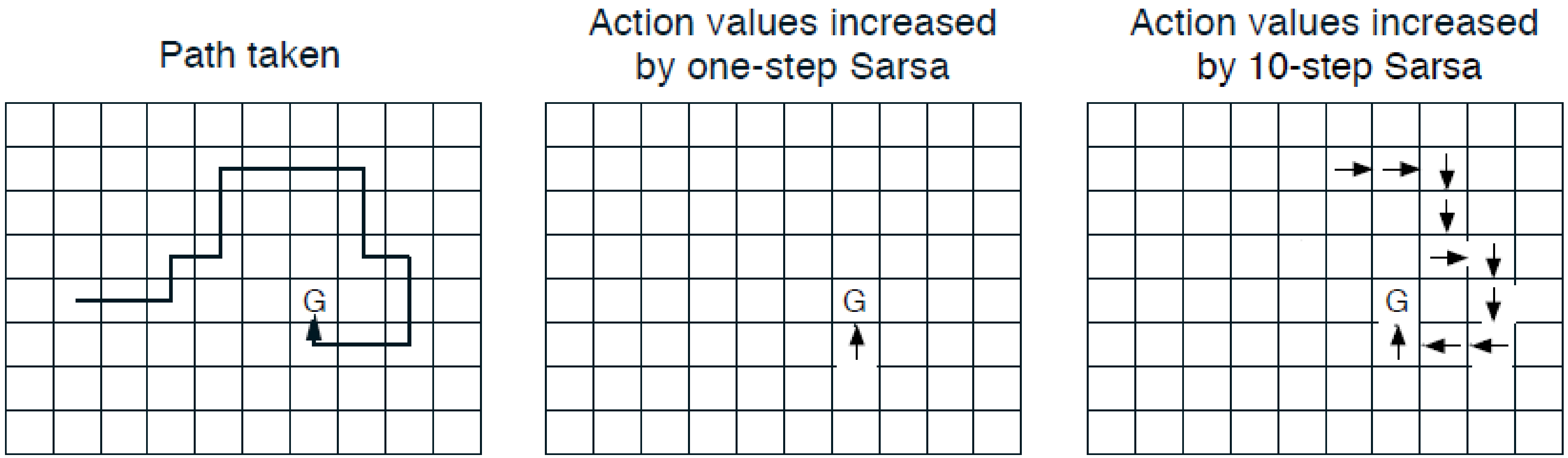


Action values increased by one-step Sarsa



Action values increased by 10-step Sarsa

Example for speed up learning



ELTE | FACULTY OF INFORMATICS

Thank you for your attention!