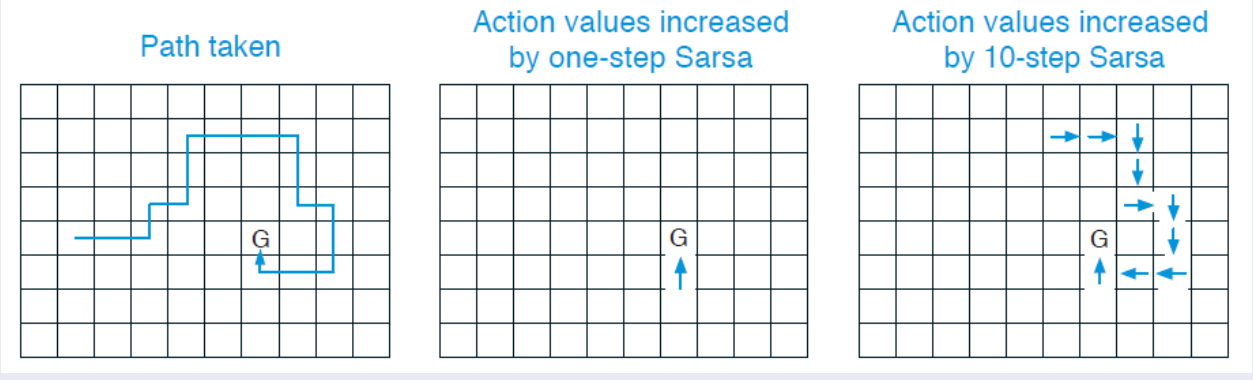
Lookahead methods

Nicolas Wicker

Laboratoire Paul Painlevé, Université Lille, 59655, Villeneuve d'Ascq, France.

The benefits of looking ahead

Time gain, illustrated here



n-step Sarsa(1/4)

for each episode **do**
  select and store action $A_0 \sim \pi(.|S_0)$
  $T \leftarrow \infty, t \leftarrow 0$
  repeat
    if $t < T$ **then**
      take action A_t
      get reward R_{t+1} and next state S_{t+1}
      if S_{t+1} is terminal **then**
        $T \leftarrow t+1$
      else
        select and store action $A_{t+1} \sim \pi(.|S_{t+1})$
      end if
    end if
    $\tau \leftarrow t-n+1$
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      if $\tau+n < T$ **then**
        $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$
      end if
      $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\big[G - Q(S_\tau, A_\tau)\big]$
    end if
    $t \leftarrow t+1$
  until $\tau = T+1$
end for

n-step Sarsa (2/4)

Some code

```
import gym;import random;import numpy;import math
env = gym.make('FrozenLake-v0')
epsilon=0.1;gamma=0.9;alpha=0.01
Q = numpy.zeros([env.observation_space.n, env.action_space.n])
n=5
nbEpisodes=30000
for i in range(0, nbEpisodes):
    statesHistory=[]
    rewardsHistory=[]
    actionsHistory=[]

    state=env.reset()
    if i%100 == 0:
        print(i)

    firstState=state
    statesHistory.append(state)
    rewardsHistory.append(0)

    endOfEpisode = False
    t=0
    tau=0
    T=math.inf
    if random.uniform(0, 1) < epsilon:
        action = env.action_space.sample()
    else:
        action = numpy.argmax(Q[state])
        actionsHistory.append(action)
```

n-step Sarsa (3/4)

Some code

```
while tau<(T-1):
    if t<T:
        next_state, reward, endOfEpisode, info = env.step(action)
        statesHistory.append(next_state)
        rewardsHistory.append(reward)
        if endOfEpisode:
            T=t+1
        else:
            if random.uniform(0, 1) < epsilon:
                action = env.action_space.sample()
            else:
                action = numpy.argmax(Q[state])
            actionsHistory.append(action)
    tau=t-n+1
    if tau>=0:
        G=0
        for j in range(tau+1,min(tau+n,T)+1):
            G=G+math.pow(gamma,j-tau-1)*rewardsHistory[j]
        if (tau+n)<T:
            G=G+math.pow(gamma,n)*Q[statesHistory[tau+n],actionsHistory[tau+n]]

        stateTau=statesHistory[tau]
        actionTau=actionsHistory[tau]
        Q[stateTau,actionTau] = (1 - alpha) * Q[stateTau,actionTau] + alpha * G
    state = next_state
    t=t+1
```

n-step Sarsa (4/4)

Results on Frozen Lake environment

- $n = 5$
- 30000 episodes
- 20% number of times the agent finds the goal

Dyna-Q (1/6)

Ideas behind Dyna-Q

- looking ahead
- planning and learning
- using previous experiences (uniform rarndom selection)

Dyna-Q (2/6)

Require: parameter n

initialize $Q(s,a)$ and Model(s,a) for $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

for $t \in 1, \ldots, \infty$ **do**

$S \leftarrow$ current state

$A \leftarrow \epsilon$ – greedy

take action A, receive reward R and change current state to S'

$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Model$(S,A) \leftarrow R, S'$

for n times **do**

$S, A \leftarrow$ random previously observed state-action pair

$R, S' \leftarrow$ Model(S,A)

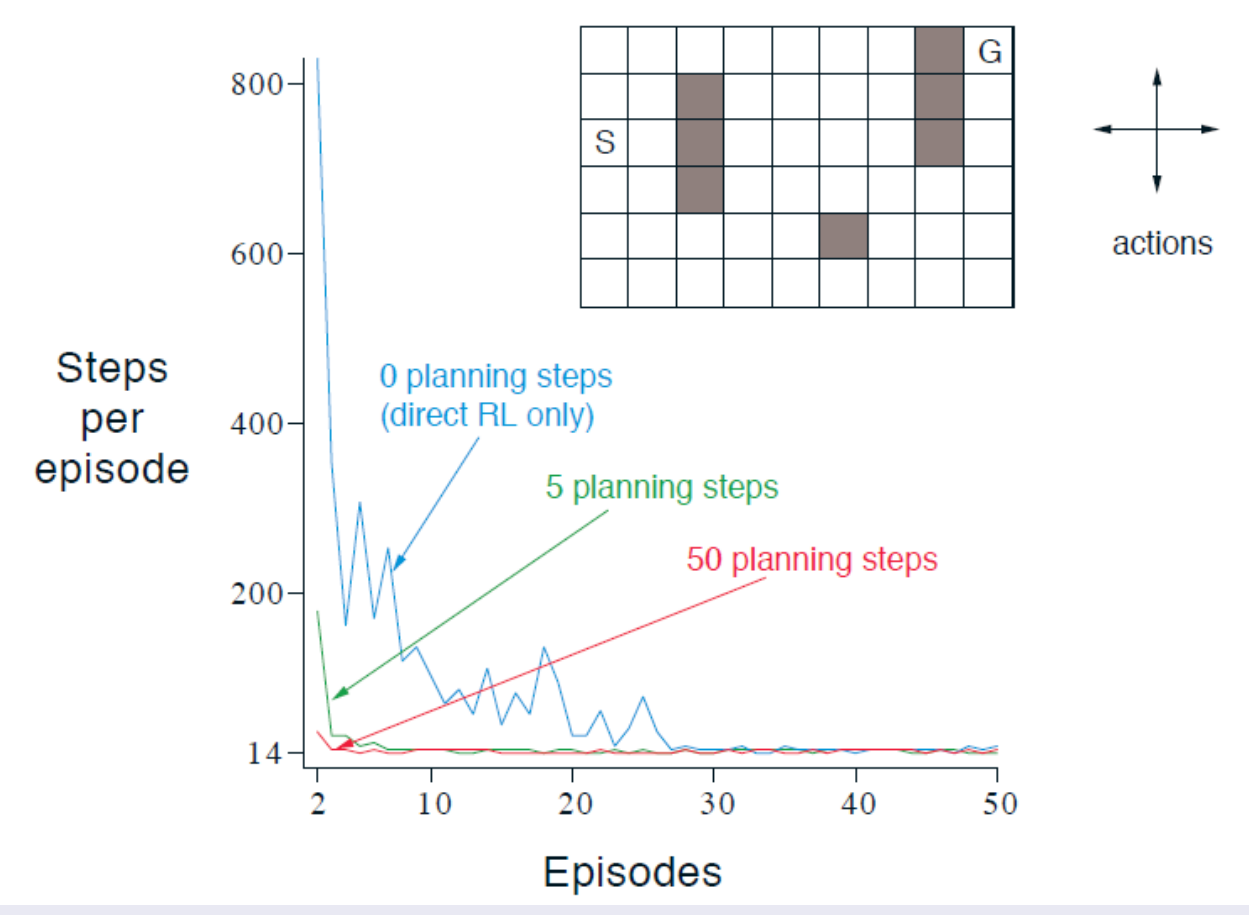
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

end for

end for

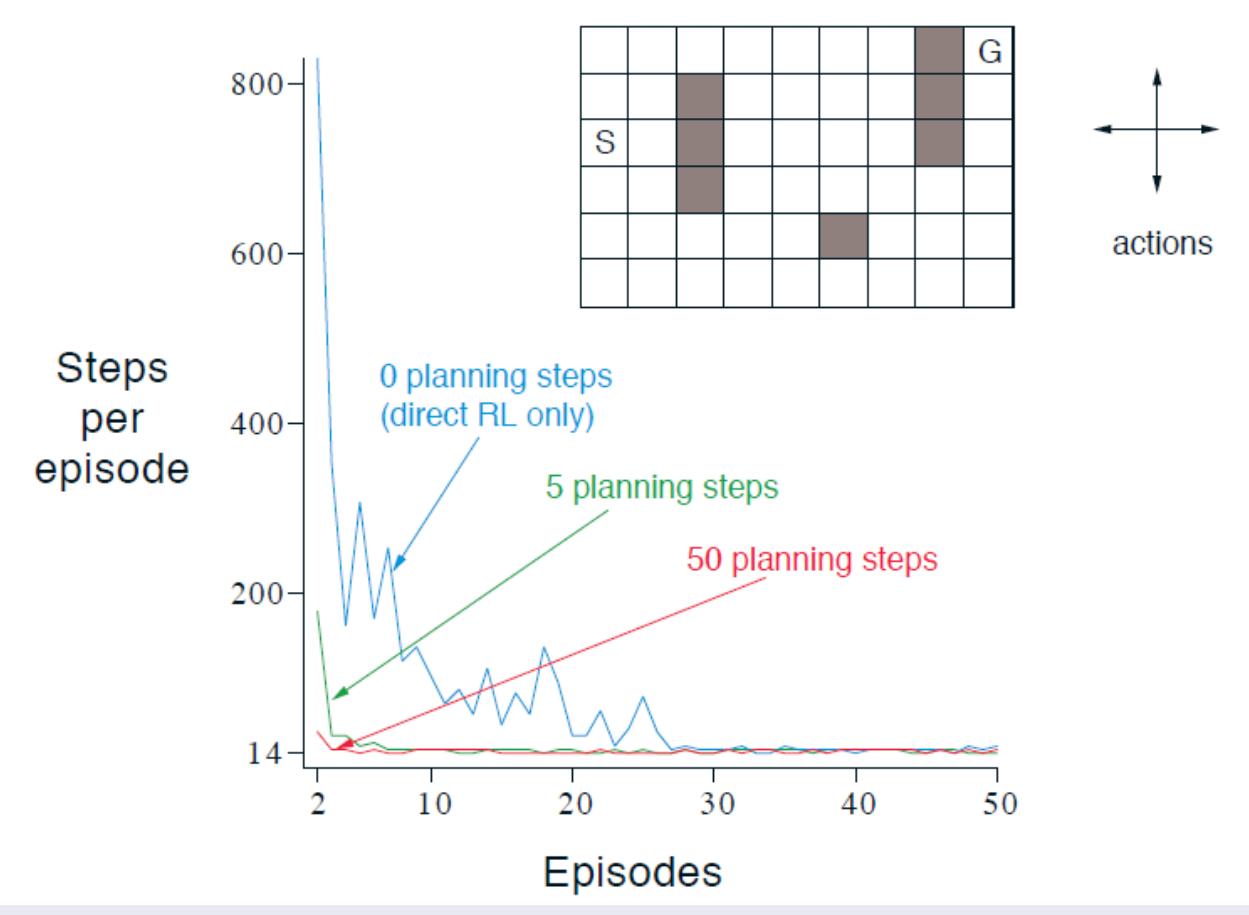
Dyna-Q (3/6)

Results



Dyna-Q (3/6)

Results



Dyna-Q (4/6)

Some code

```
import gym;import random;import numpy
env = gym.make('FrozenLake-v0')
epsilon=0.1;gamma=0.9;alpha=0.01

Q = numpy.zeros([env.observation_space.n, env.action_space.n])
model=[]
for i in range(0,env.observation_space.n):
    model.append(0)
encounteredStates=[]

nDynaQ=5
nbEpisodes=50000
for i in range(0, nbEpisodes):
    state=env.reset()
    firstState=state
    endOfEpisode = False
    nbSteps=0
```

Dyna-Q (5/6)

Some code

```
while not endOfEpisode:
    if random.uniform(0, 1) < epsilon:
        action = env.action_space.sample()
    else:
        action = numpy.argmax(Q[state])

    next_state, reward, endOfEpisode, info = env.step(action)
    model[state]=[action,reward,next_state]
    if encounteredStates.count(state)==0:
        encounteredStates.append(state)

    next_max = numpy.max(Q[next_state])

    Q[state, action] = (1 - alpha) * Q[state,action] + alpha * (reward + gamma * next_max)

    for j in range(0,nDynaQ):
        selectedState=random.sample(encounteredStates,1)[0]
        [action,R,Sprime]=model[selectedState]
        Q[selectedState,action]=(1-alpha)*Q[selectedState,action]+ \
        alpha*(R+gamma*numpy.max(Q[Sprime]))

    state = next_state
    nbSteps=nbSteps+1
```

Dyna-Q (6/6)

Results on Frozen Lake environment

- 50000 iterations
- $n = 5$
- 73.9% success

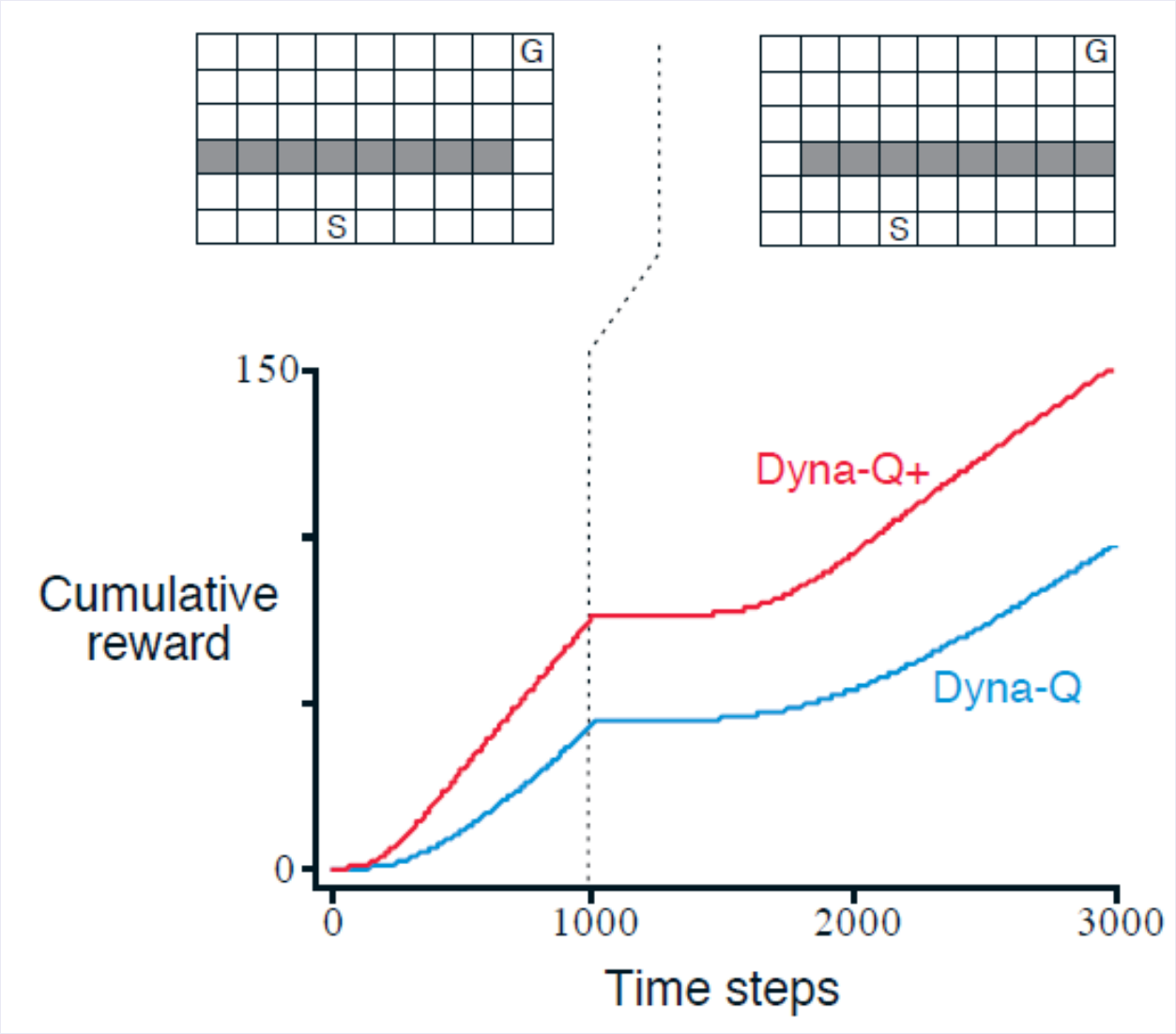
Dyna-Q+ (1/2)

Slight variation over Dyna algorithm

- if the model is wrong or changes then DynaQ may take time to notice or even not notice it at all
- long untried state-action pairs receive a bonus reward like $r + \kappa\sqrt{\tau}$
- then new solutions can be found

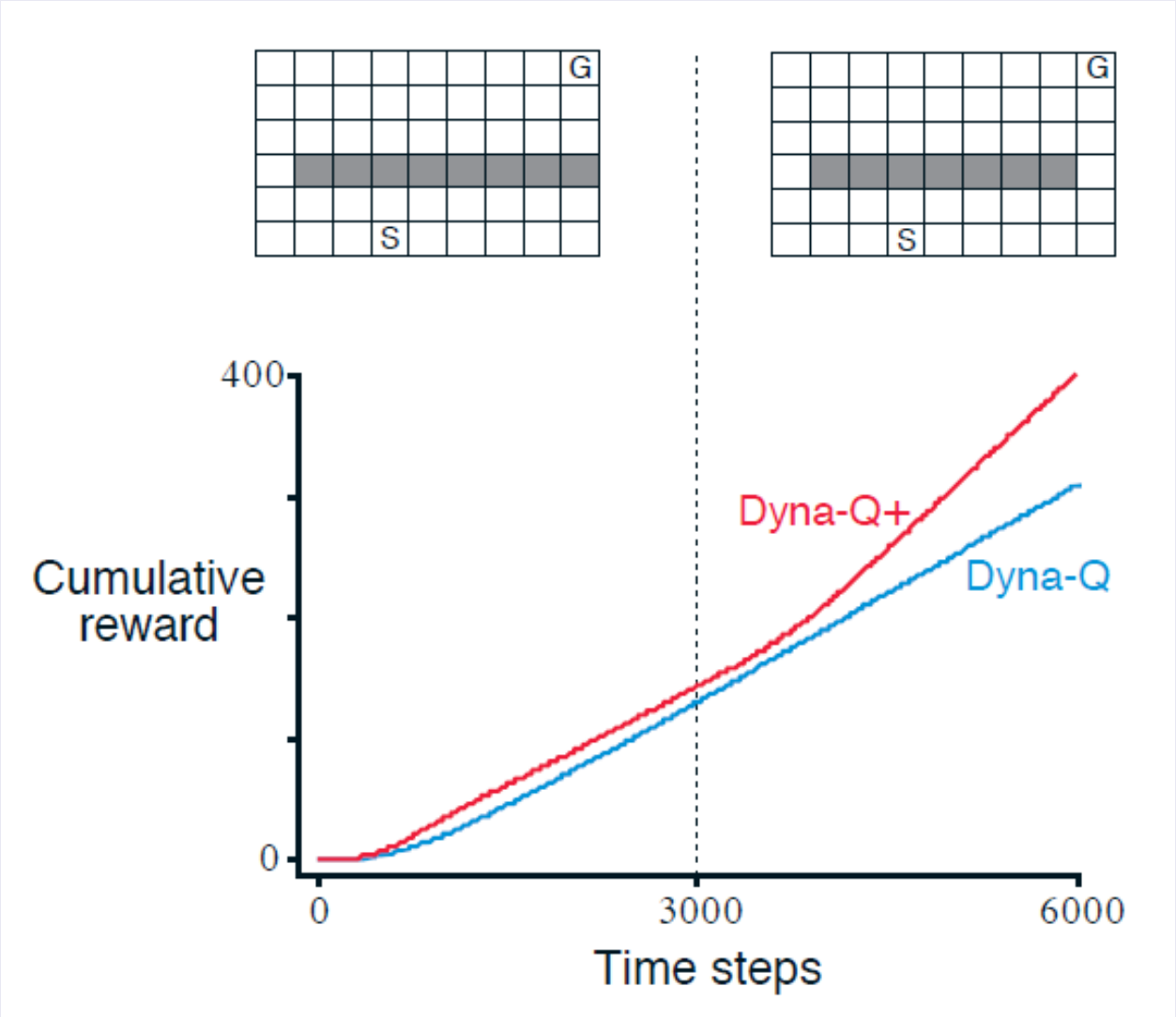
Dyna-Q+ (2/2)

Results



Dyna-Q+ (2/2)

Results



Prioritized sweeping (1/3)

Motivation

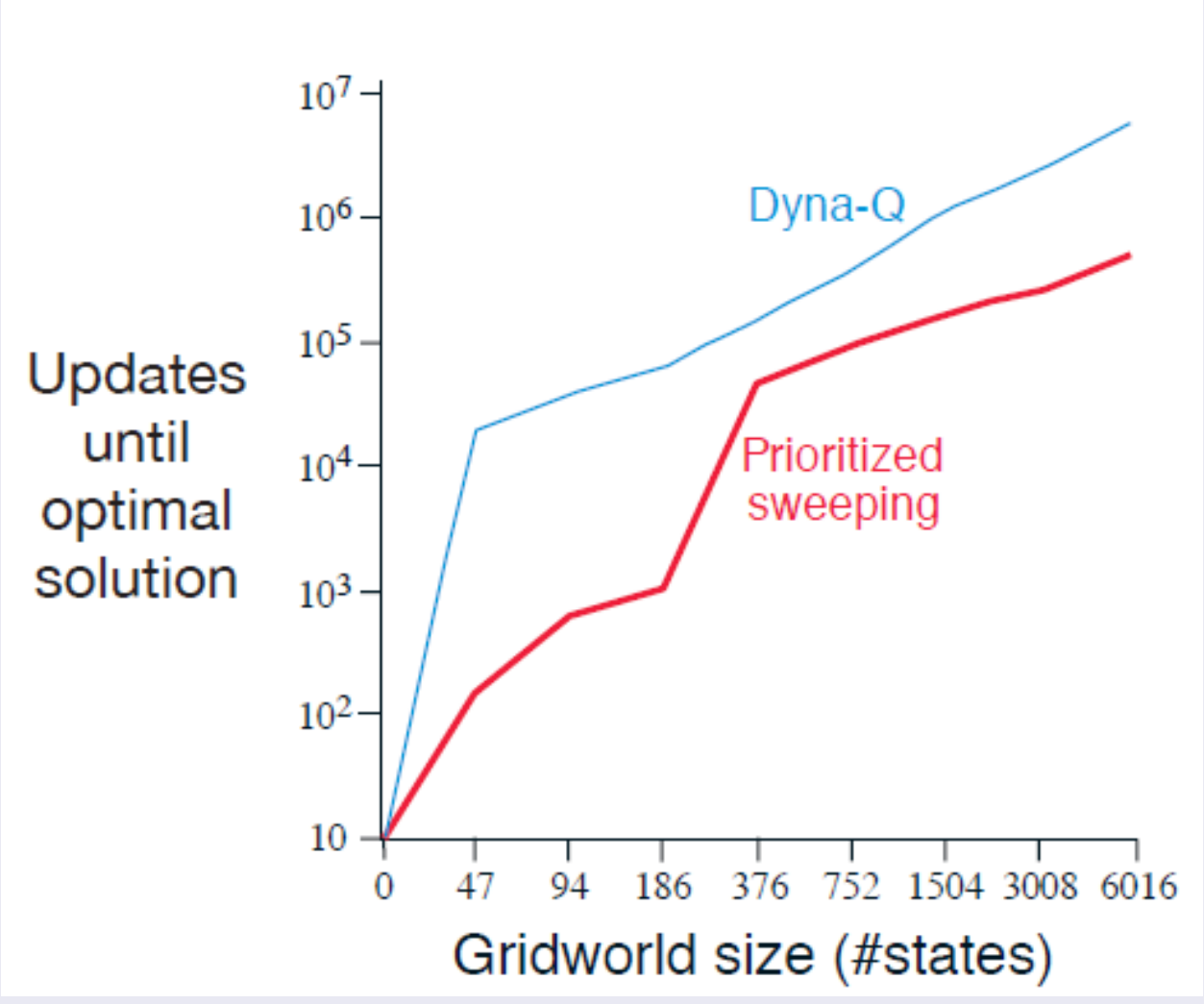
- Dyna selects state-action pairs uniformly, which may be inefficient
- better to start from the "goal" states and proceed backward

Prioritized sweeping (2/3)

$H_1 \leftarrow H$
for $t \in 1, \ldots, \infty$ **do**
  receive $A \leftarrow \pi(S, Q)$
  observe resulting reward R and state S'
  $\mathrm{Model}(S, A) \leftarrow R, S'$
  $P \leftarrow \left|R + \gamma \max_a Q(S', a) - Q(S, A)\right|$
  if $P > \eta$ then insert S, A into PQueue with priority P
  for $i \in 1, \ldots, n$ **do**
    $S, A \leftarrow \mathrm{first}(\mathrm{PQueue})$
    $R, S' \leftarrow \mathrm{Model}(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    for all $\bar{S}, \bar{A}$ predicted to lead to S **do**
      $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
      $P \leftarrow \left|\bar{R} + \gamma \max_a Q(S, a) - Q(\bar{S}, \bar{A})\right|$
      if $P > \eta$ then insert $\bar{S}, \bar{A}$ into PQueue with priority P
    end for
  end for
end for

Prioritized sweeping (3/3)

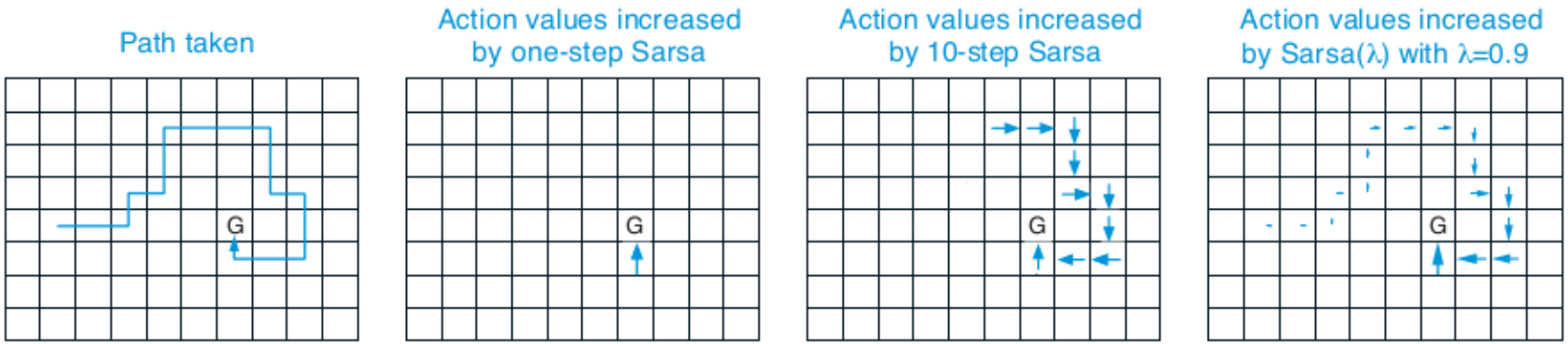
Results



TD(λ) (1/2)

Motivation

- like prioritize sweeping, the aim is to look several steps ahead
- the strategy relies here on fading old steps



TD(λ) (2/2)

$V \leftarrow V_0, e \leftarrow 0$
for each episode **do**
  for each step of current episode **do**
    take action A, observe R, S'
    $\delta \leftarrow R + \lambda V(S') - V(S)$
    $e(S) \leftarrow \lambda e(s) + 1$
    for $u \in \mathcal{S}$ **do**
      if $u \neq s$ **then**
        $e(u) \leftarrow \gamma\lambda e(u)$
      end if
      $V(u) \leftarrow V(u) + \alpha\delta e(u)$
    end for
  end for
end for