Chapter 9: Planning and Learning

Objectives of this chapter:

- Use of environment models
- Integration of planning and learning methods

Models

- Model: anything the agent can use to predict how the environment will respond to its actions
- Distribution model: description of all possibilities and their probabilities
  - e.g., $P_{ss'}^{a}$ and $R_{ss'}^{a}$ for all $s,\ s',$ and $a \in A(s)$
- Sample model: produces sample experiences
  - e.g., a simulation model
- Both types of models can be used to produce simulated experience
- Often sample models are much easier to come by

Planning

- Planning: any computational process that uses a model to create or improve a policy

$$\text{model} \xrightarrow{\text{planning}} \text{policy}$$

- Planning in AI:
  - state-space planning
  - plan-space planning (e.g., partial-order planner)
- We take the following (unusual) view:
  - all state-space planning methods involve computing value functions, either explicitly or implicitly
  - they all apply backups to simulated experience

$$\text{model} \longrightarrow \begin{array}{c}\text{simulated}\\ \text{experience}\end{array} \xrightarrow{\text{backups}} \text{values} \longrightarrow \text{policy}$$

Planning Cont.

- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- A planning method based on Q-learning:

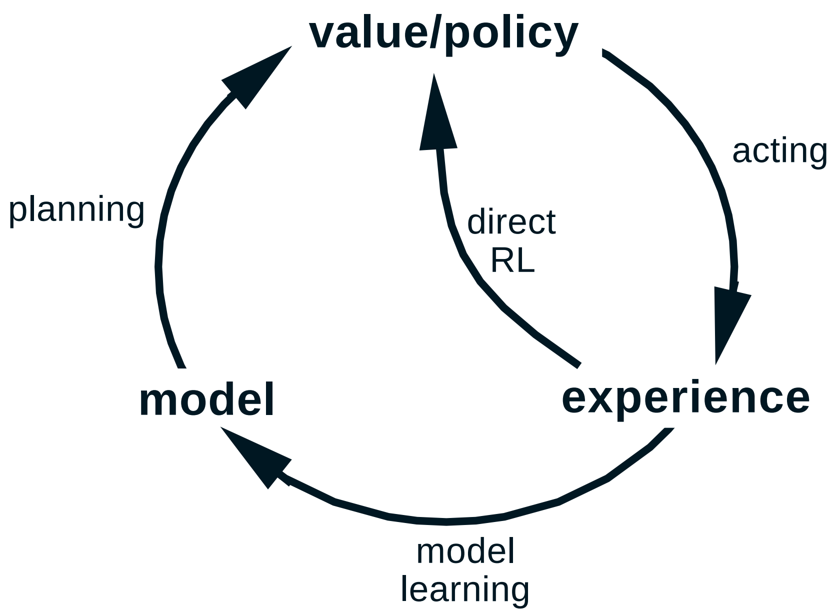
Do forever:

1. Select a state, $s \in \mathcal{S}$, and an action, $a \in \mathcal{A}(s)$, at random
2. Send s, a to a sample model, and obtain a sample next state, s', and a sample next reward, r
3. Apply one-step tabular Q-learning to s, a, s', r:
   $$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Random-Sample One-Step Tabular Q-Planning

Learning, Planning, and Acting

- Two uses of real experience:
  - model learning: to improve the model
  - direct RL: to directly improve the value function and policy
- Improving value function and/or policy via a model is sometimes called indirect RL or model-based R L. Here, we call it planning.



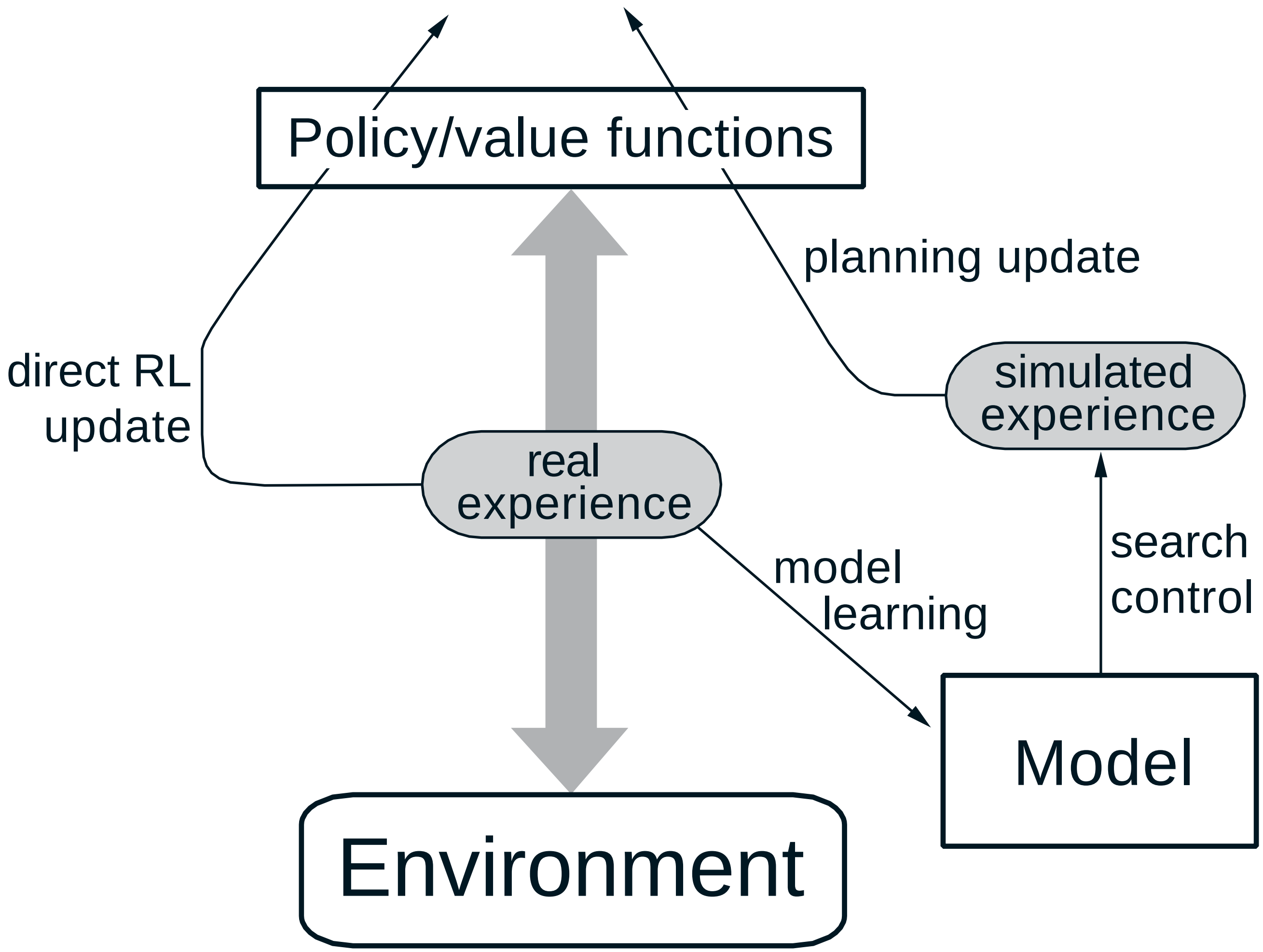
Direct vs. Indirect RL

- Indirect methods:
  - make fuller use of experience: get better policy with fewer environment interactions
- Direct methods
  - simpler
  - not affected by bad models

But they are very closely related and can be usefully combined:

planning, acting, model learning, and direct RL can occur simultaneously and in parallel

The Dyna Architecture (Sutton 1990)



The Dyna-Q Algorithm

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

Do forever:

(a) $s \leftarrow$ current (nonterminal) state

(b) $a \leftarrow \epsilon$-greedy(s, Q)

(c) Execute action a; observe resultant state, s', and reward, r

(d) $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$ ← direct RL

(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment) ← model learning

(f) Repeat N times:

$s \leftarrow$ random previously observed state

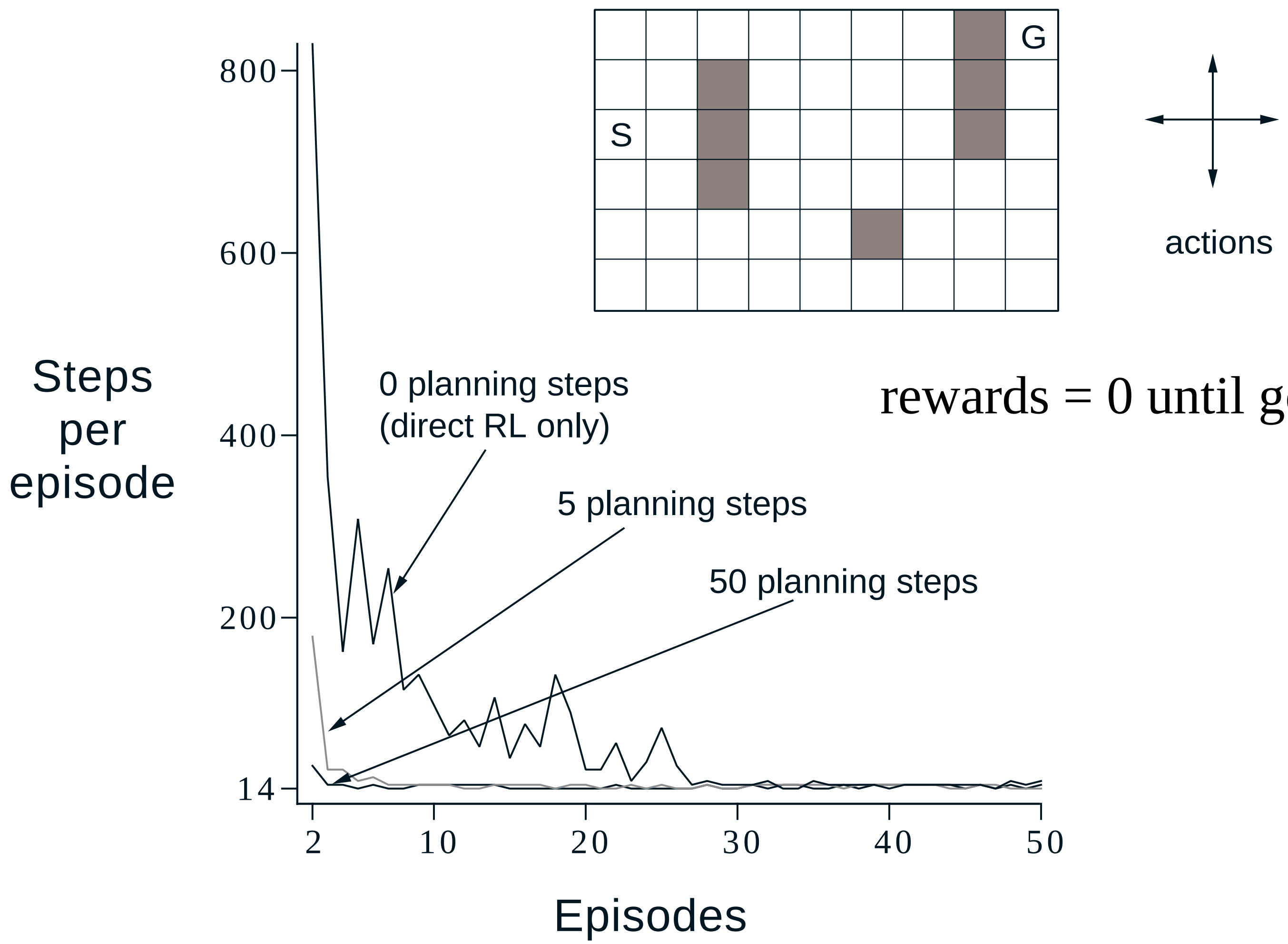
$a \leftarrow$ random action previously taken in s

$s', r \leftarrow Model(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

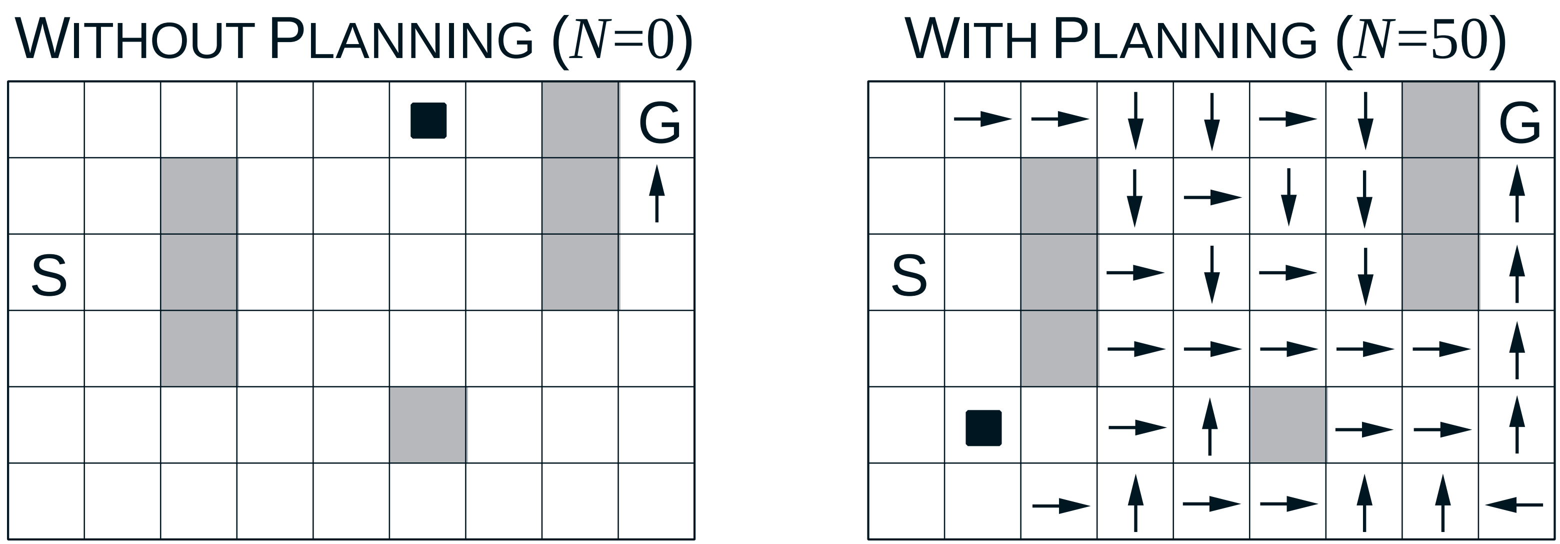
← planning

Dyna-Q on a Simple Maze



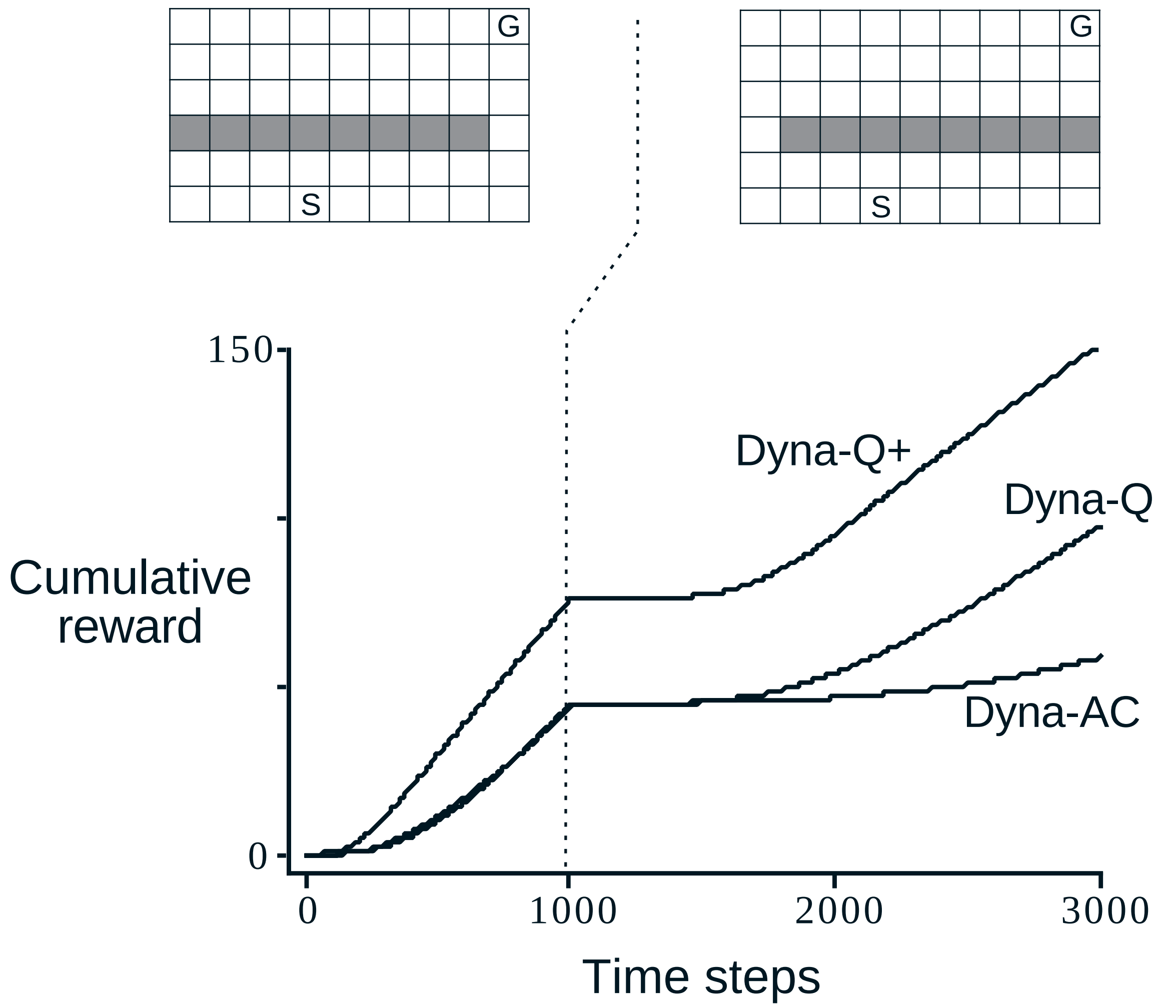
rewards = 0 until goal, when =1

Dyna-Q Snapshots: Midway in 2nd Episode



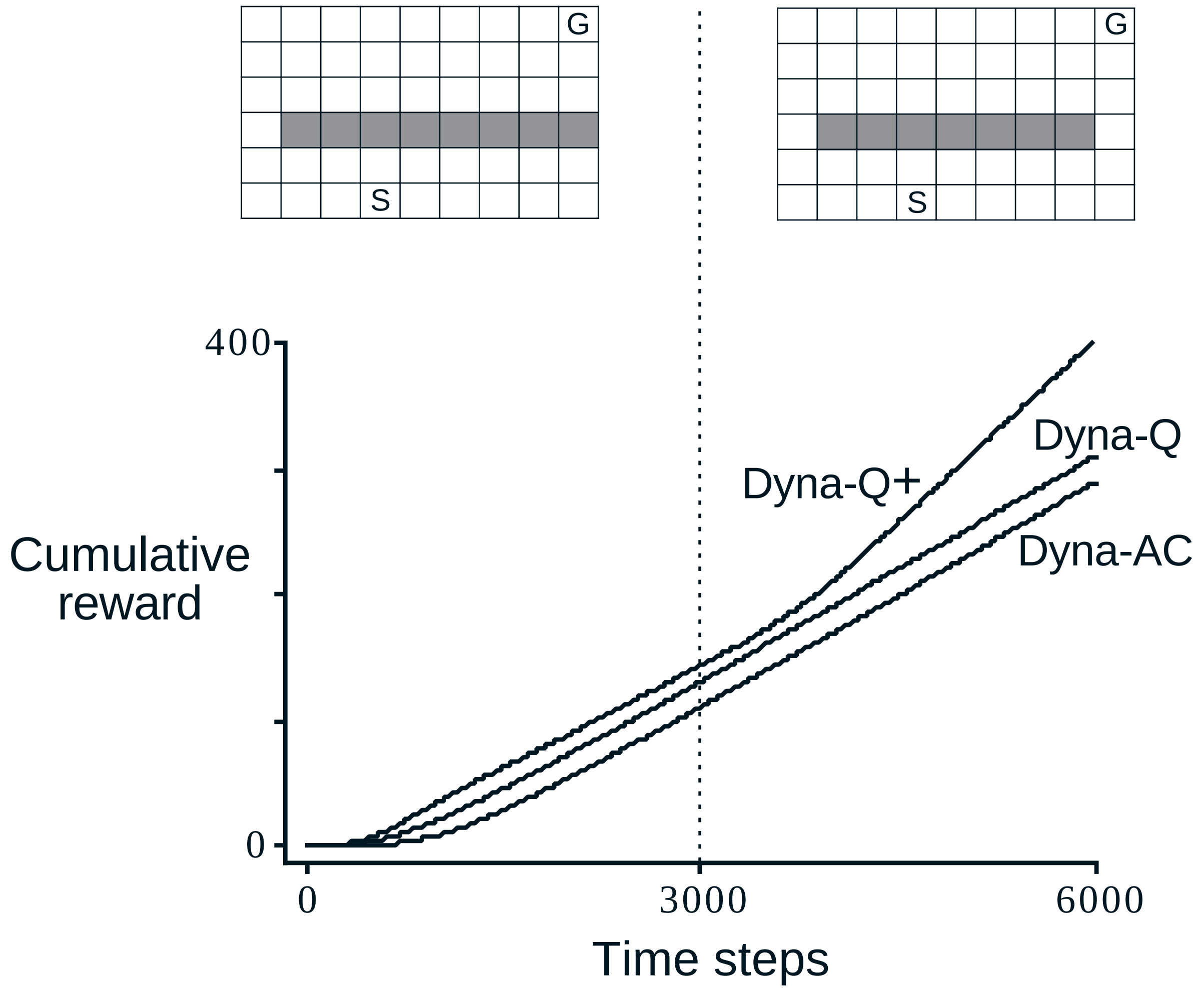
When the Model is Wrong: Blocking Maze

The changed envirnoment is harder



Shortcut Maze

The changed environment is easier



What is Dyna-Q$^+$?

- Uses an “exploration bonus”:
  - Keeps track of time since each state-action pair was tried for real
  - An extra reward is added for transitions caused by state-action pairs related to how long ago they were tried: the longer unvisited, the more reward for visiting
  - The agent actually “plans” how to visit long unvisited states

Prioritized Sweeping

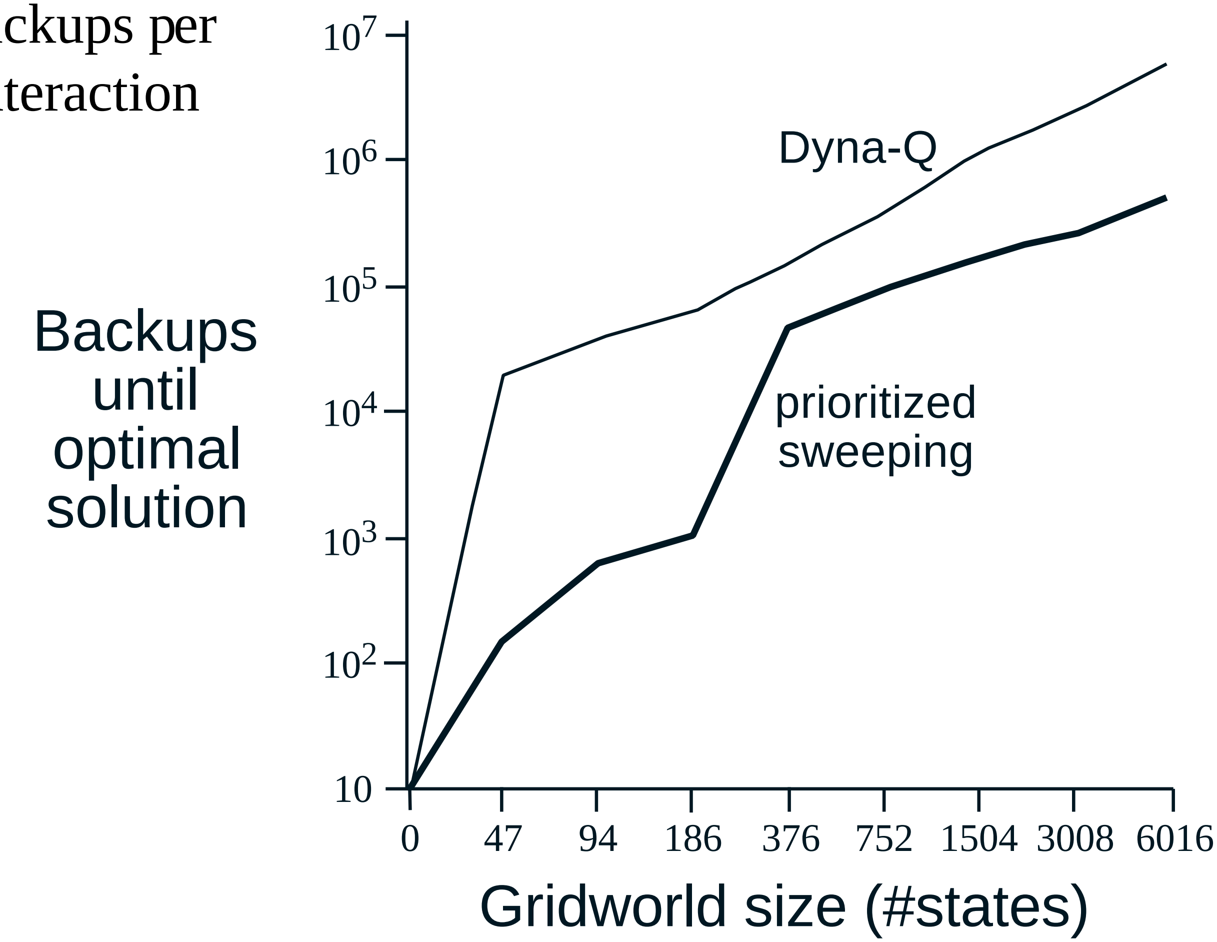
- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
  - Maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
  - When a new backup occurs, insert predecessors according to their priorities
  - Always perform backups from first in queue
- Moore and Atkeson 1993; Peng and Williams, 1993

Prioritized Sweeping

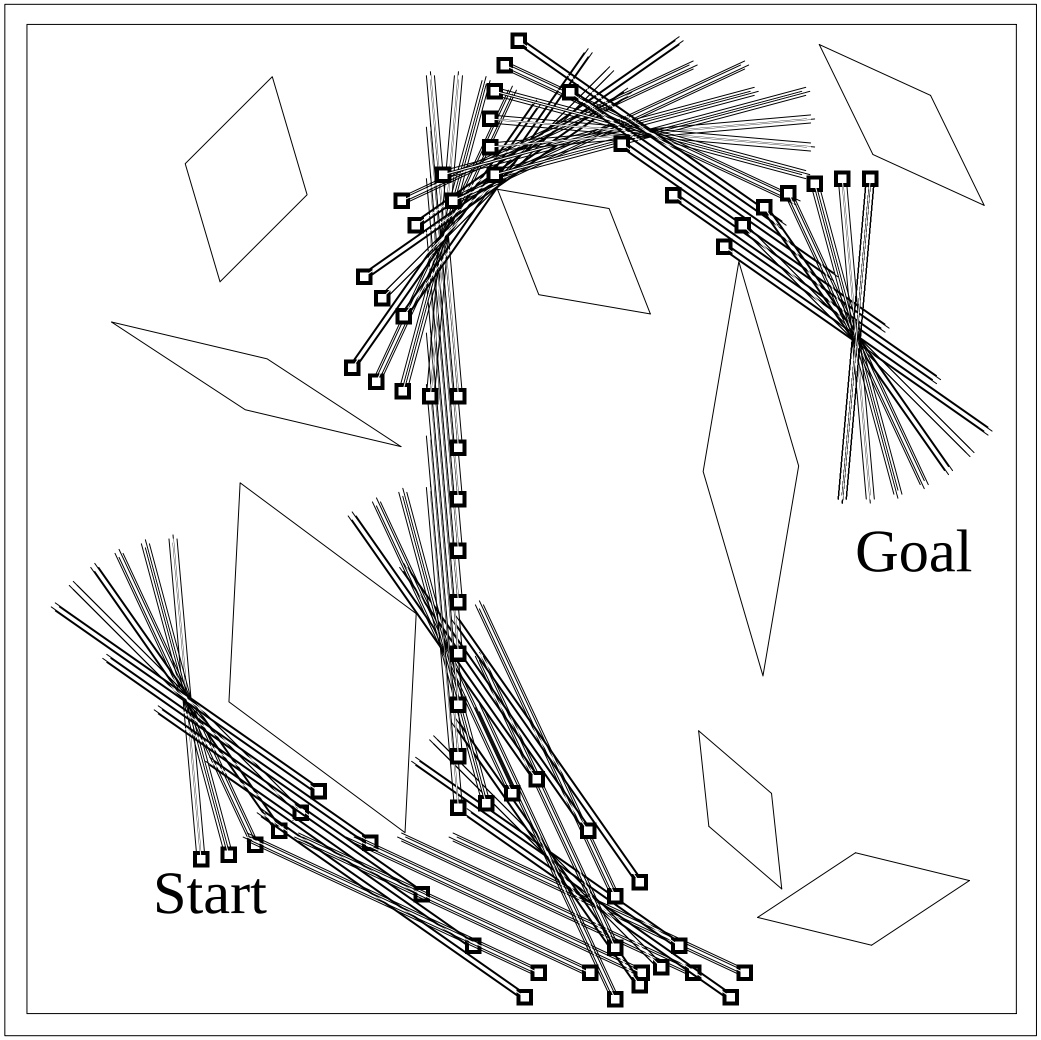
Initialize $Q(s,a)$, $Model(s,a)$, for all s, a, and $PQueue$ to empty
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow policy(s, Q)$
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Model(s,a) \leftarrow s', r$
(e) $p \leftarrow |r + \gamma \max_{a'} Q(s',a') - Q(s,a)|$.
(f) if $p > \theta$, then insert s, a into $PQueue$ with priority p
(g) Repeat N times, while $PQueue$ is not empty:
    $s, a \leftarrow \text{first}(PQueue)$
    $s', r \leftarrow Model(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    Repeat, for all $\bar{s}, \bar{a}$ predicted to lead to s:
        $\bar{r} \leftarrow$ predicted reward
        $p \leftarrow |\bar{r} + \gamma \max_a Q(s,a) - Q(\bar{s},\bar{a})|$.
        if $p > \theta$ then insert $\bar{s}, \bar{a}$ into $PQueue$ with priority p

Prioritized Sweeping vs. Dyna-Q

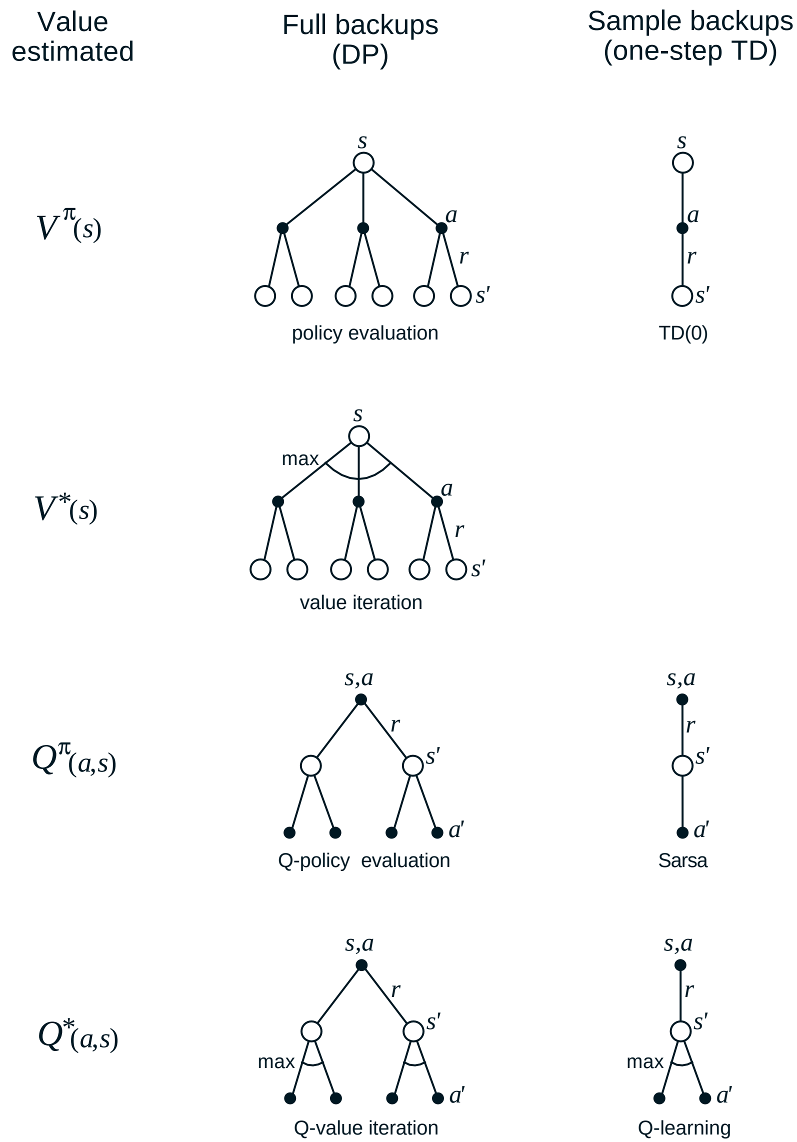
Both use N=5 backups per environmental interaction



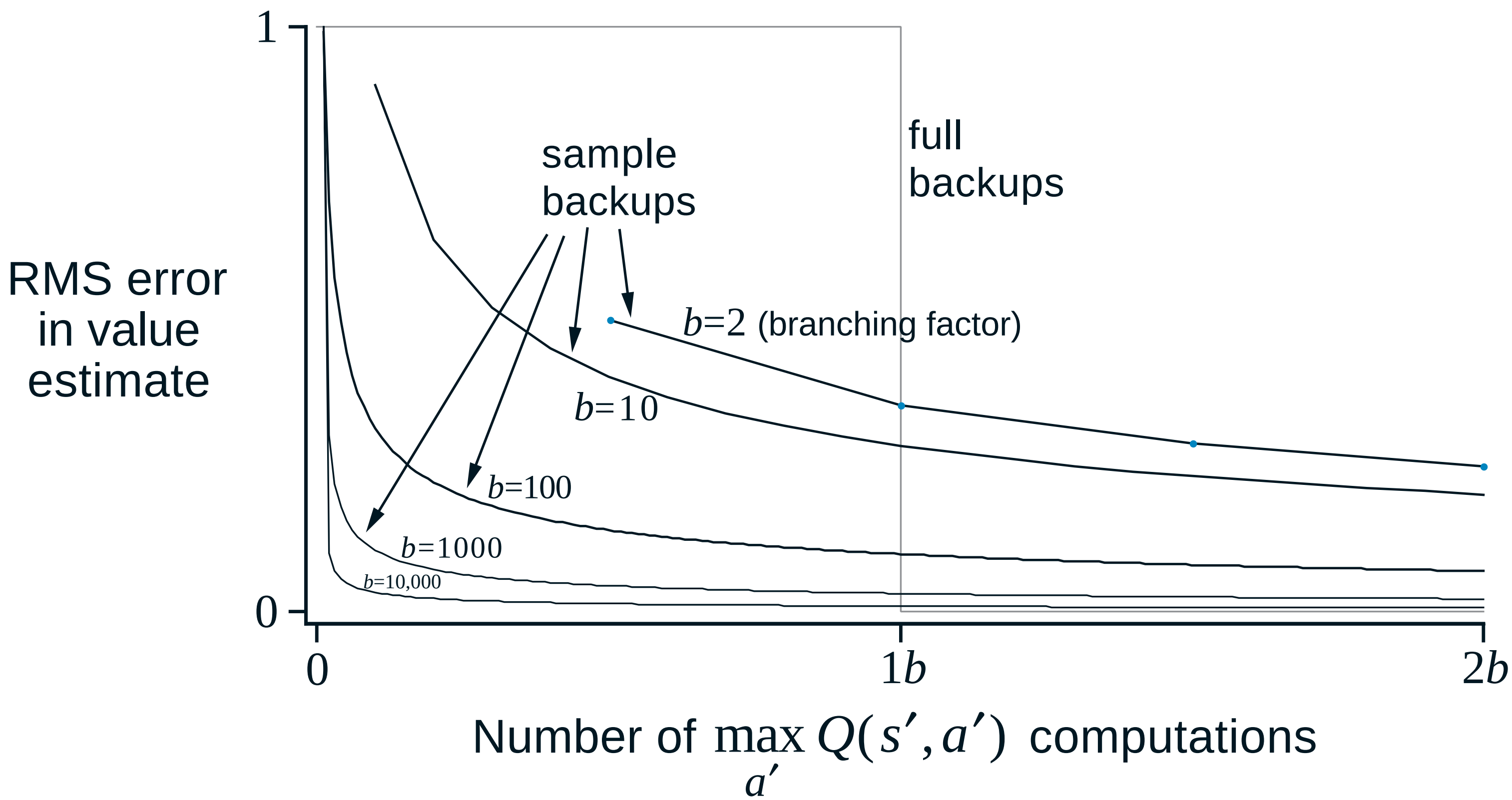
Rod Maneuvering (Moore and Atkeson 1993)



Full and Sample (One-Step) Backups



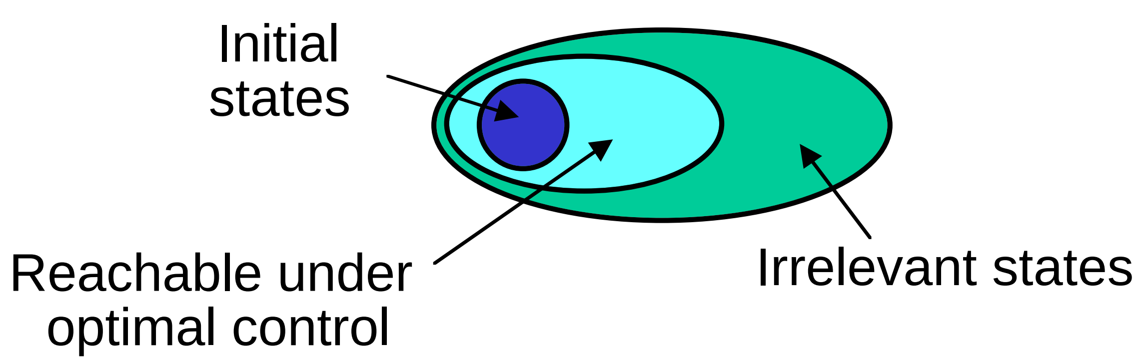
Full vs. Sample Backups



b successor states, equally likely; initial error = 1;
assume all next states' values are correct

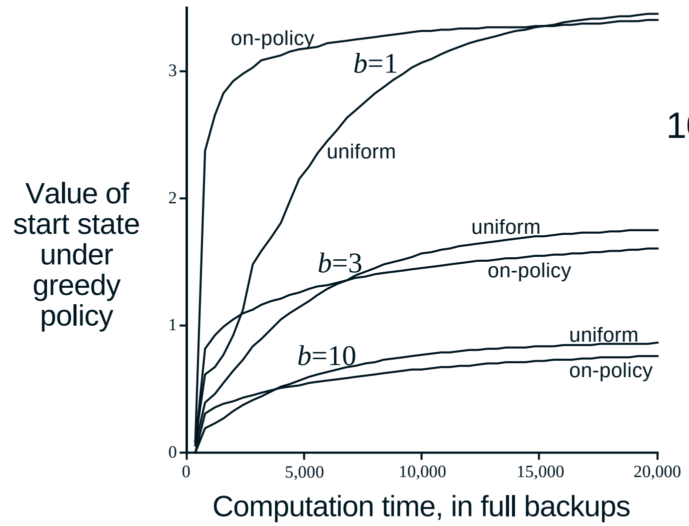
Trajectory Sampling

- Trajectory sampling: perform backups along simulated trajectories
- This samples from the on-policy distribution
- Advantages when function approximation is used (Chapter 8)
- Focusing of computation: can cause vast uninteresting parts of the state space to be (usefully) ignored:

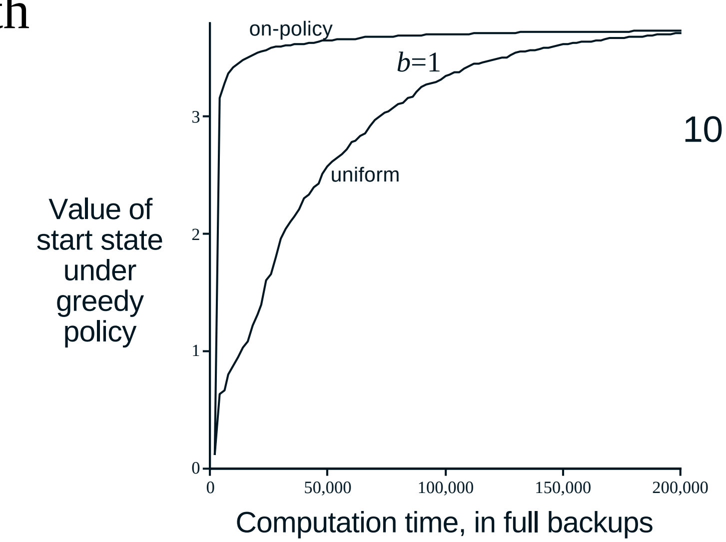


Trajectory Sampling Experiment

- one-step full tabular backups
- uniform: cycled through all state-action pairs
- on-policy: backed up along simulated trajectories
- 200 randomly generated undiscounted episodic tasks
- 2 actions for each state, each with b equally likely next states
- .1 prob of transition to terminal state
- expected reward on each transition selected from mean 0 variance 1 Gaussian



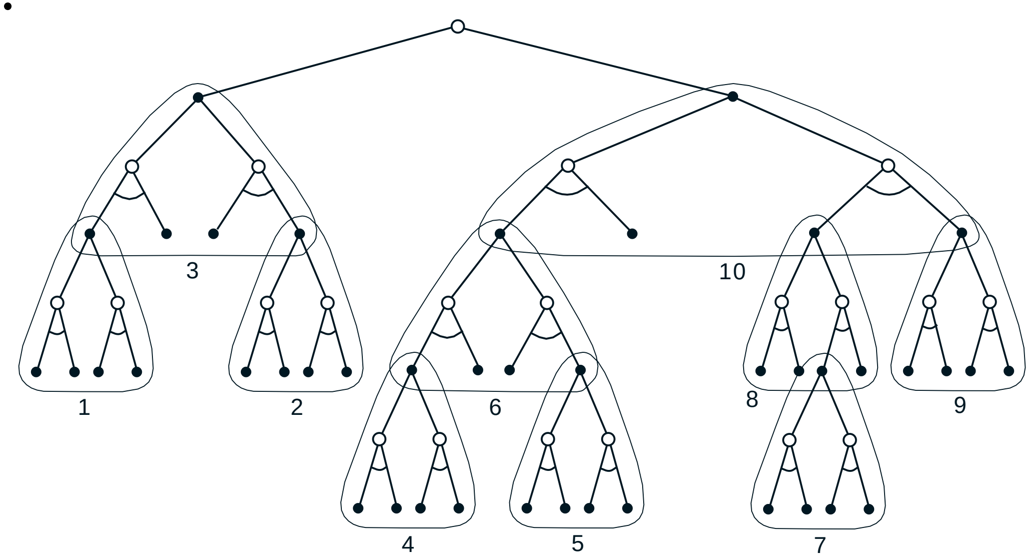
1000 STATES



10,000 STATES

Heuristic Search

- Used for action selection, not for changing a value function (=heuristic evaluation function)
- Backed-up values are computed, but typically discarded
- Extension of the idea of a greedy policy — only deeper
- Also suggests ways to select states to backup: smart focusing:



Summary

- Emphasized close relationship between planning and learning
- Important distinction between distribution models and sample models
- Looked at some ways to integrate planning and learning
  - synergy among planning, acting, model learning
- Distribution of backups: focus of the computation
  - trajectory sampling: backup along trajectories
  - prioritized sweeping
  - heuristic search
- Size of backups: full vs. sample; deep vs. shallow