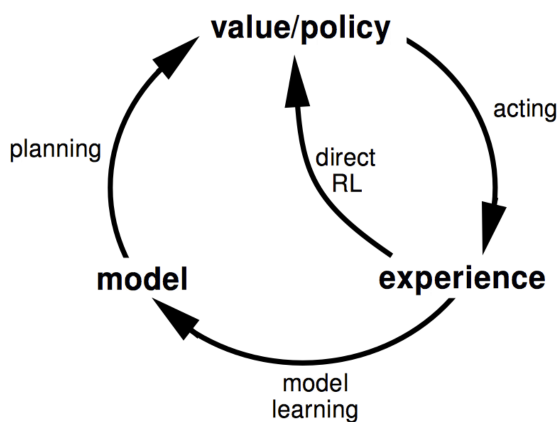
Planning and Learning

Suggested reading:

Chapter 9 in R. S. Sutton, A. G. Barto: Reinforcement Learning: An Introduction
MIT Press, 1998.

Planning and Learning



Contents:

- Models
- Planning
- Direct, indirect RL
- Dyna Architecture
- Prioritized sweeping
- Full vs sample backups
- Trajectory sampling
- Heuristic search

Models

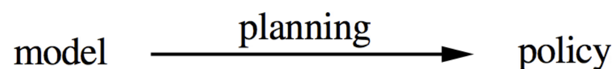
- Use of environment models
- Integration of planning and learning methods

Definitions:

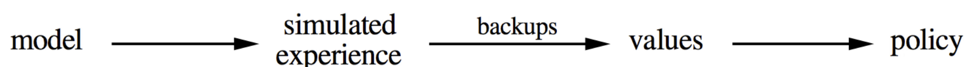
- Model: anything the agent can use to predict how the environment will respond to its actions
- Distribution model: description of all possibilities and their probabilities
  - e.g., $P^a_{ss'}$ and $R^a_{ss'}$ for all s, s', and $a \in A(s)$
- Sample model: produces sample experiences
  - e.g., a simulation 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



- Planning in AI:
  - state-space planning
  - plan-space planning (search through the space of plans, 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



Planning

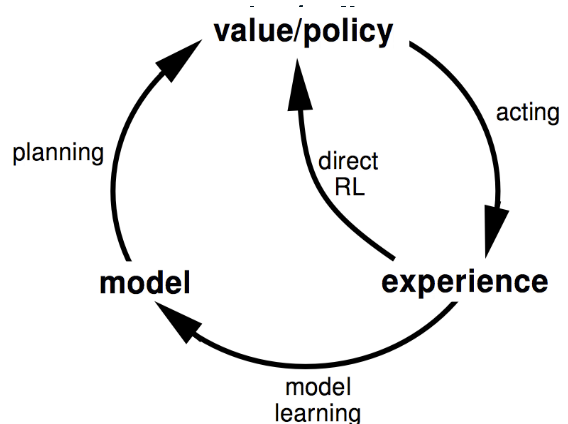
- Classical DP methods are state-space planning methods
- Heuristic search methods are state-space planning methods
- A planning method based on Q-learning:

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

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 RL. Here, we call it planning.



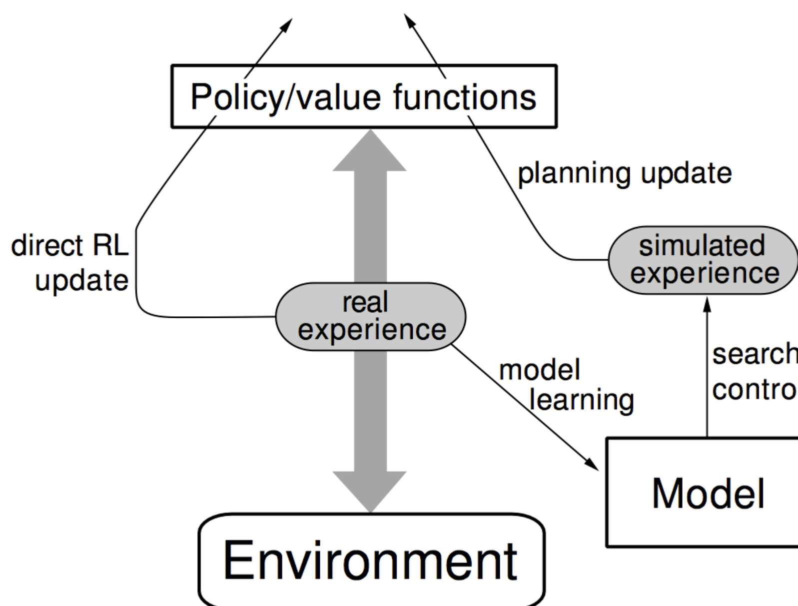
Direct vs. Indirect RL

- 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)



Planning is achieved by applying reinforcement learning methods to the simulated experiences just as if they had really happened.

Planning: random-sample one-step tabular Q-planning method.
Direct RL learning: one-step tabular Q-learning.
Model-learning: table-based assumes deterministic world, $(s_t,a_t)\text{->} (s_{t+1},r_{t+1})$

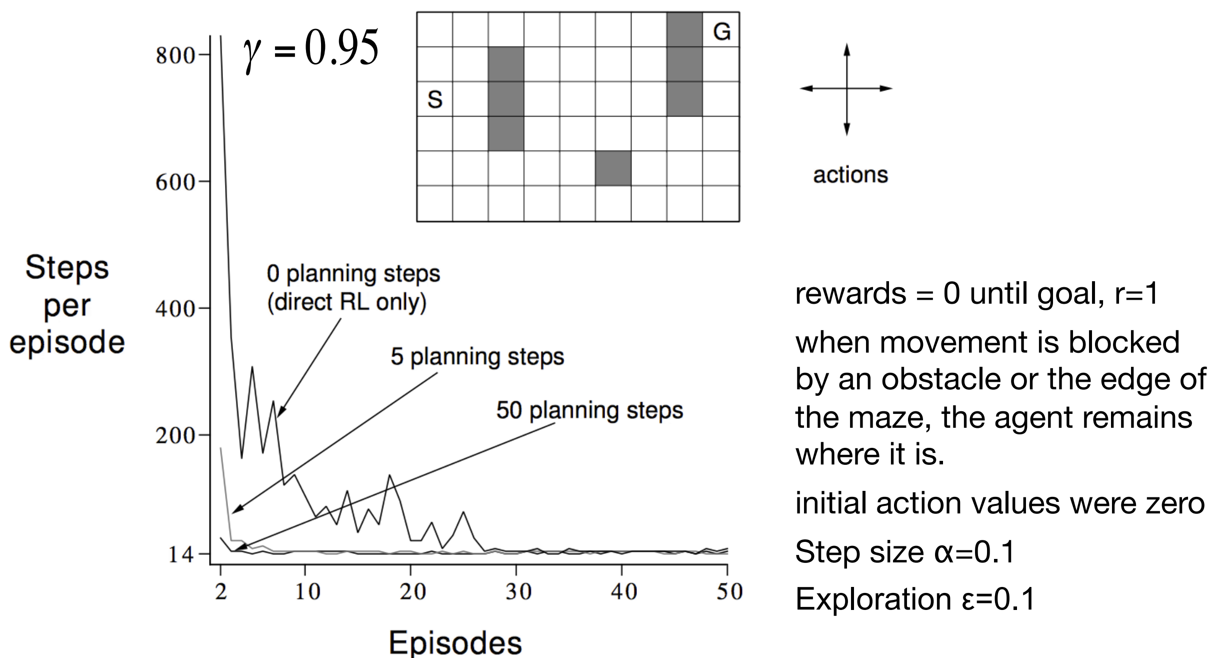
During planning, the Q-planning algorithm randomly samples only from state-action pairs that have previously been experienced, so the model is never queried with a pair about which it has no information.

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) $a \leftarrow \varepsilon$-greedy(s, Q)
(c) Execute action a; observe resultant state, s', and reward, r
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ ← **direct RL**
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment) ← **model learning**
(f) Repeat N times: ← **planning**
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in s
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

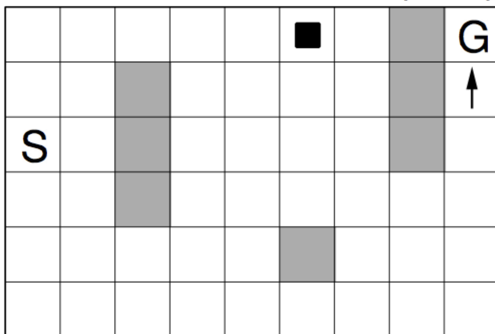
Model(s,a) denotes the contents of the model (predicted next state and reward) for state-action pair *s,a*.

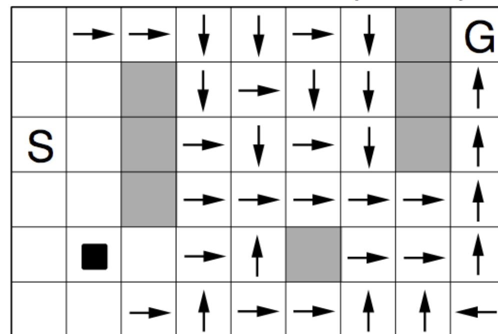
Dyna-Q on a Simple Maze



Dyna-Q on a Simple Maze

Snapshots: Midway in 2nd Episode



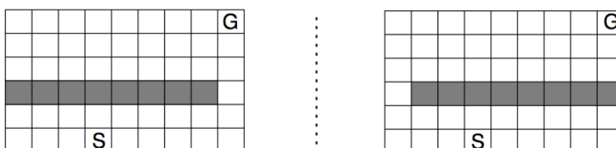


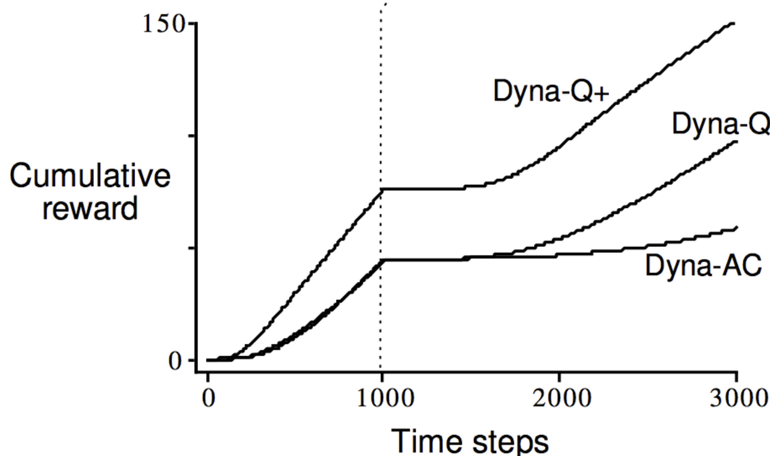
Without planning, each episode adds only one additional step to the policy, and so only one step (the last) has been learned so far.

With planning, again only one step is learned during the first episode, but here during the second episode an extensive policy has been developed by the planning process.

When the Model is Wrong

Blocking Maze: The changed environment is harder





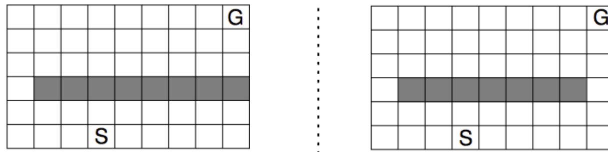
Average performance of Dyna agents on a blocking task.

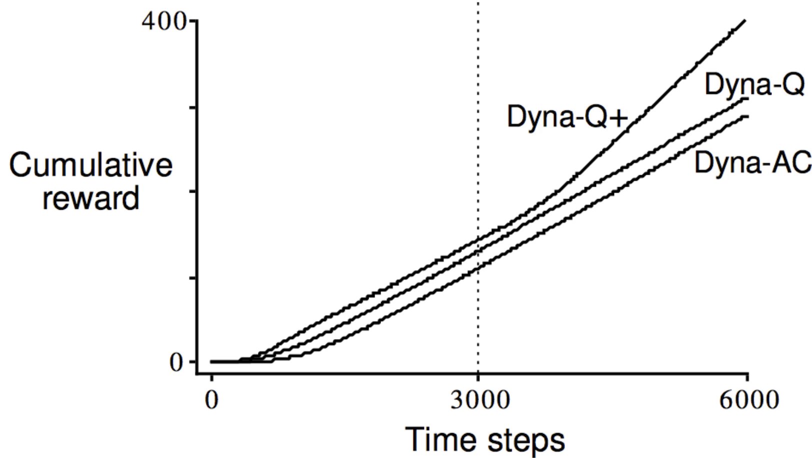
Dyna-Q+ is Dyna-Q with an exploration bonus that encourages exploration.

Dyna-AC is a Dyna agent that uses an actor-critic learning method instead of Q-learning.

When the Model is Wrong

Shortcut Maze: The changed environment is harder





Two of the three Dyna agents never switched to the shortcut.

Even with an ε-greedy policy, it is very unlikely that an agent will take so many exploratory actions as to discover the shortcut.

What is Dyna-Q+ ?

Uses an "exploration bonus":

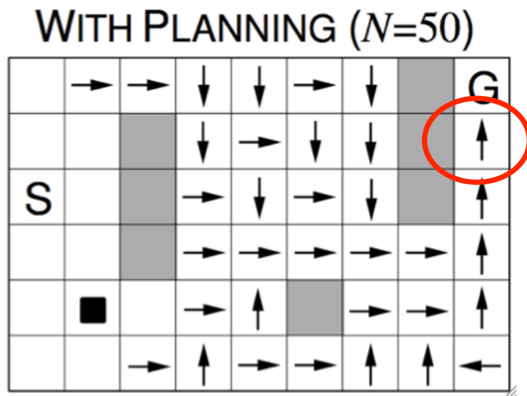
- Keeps track of time since each state-action pair was tried for real (*n*: number of time-steps a transition has not occurred)
- An extra reward is added for transitions caused by state-action pairs related to how long ago they were tried: the longer unvisited, the more reward for visiting

$$r + \kappa\sqrt{n}$$

- The agent actually "plans" how to visit long unvisited states

Prioritized Sweeping

In the Dyna agents simulated transitions are started in state-action pairs selected uniformly at random from all previously experienced pairs.
But a uniform selection is usually not the best!



Only the state-action pair leading directly into the goal has a positive value; the values of all other pairs are still zero.
It is pointless to back up along almost all transitions, because they take the agent from one zero-valued state to another, and thus the backups would have no effect.

This example suggests that search might be usefully focused by working *backward from goal states!??*
In general, we want to work back from any state whose value has changed.

Prioritized Sweeping

Prioritize the backups according to a measure of their urgency

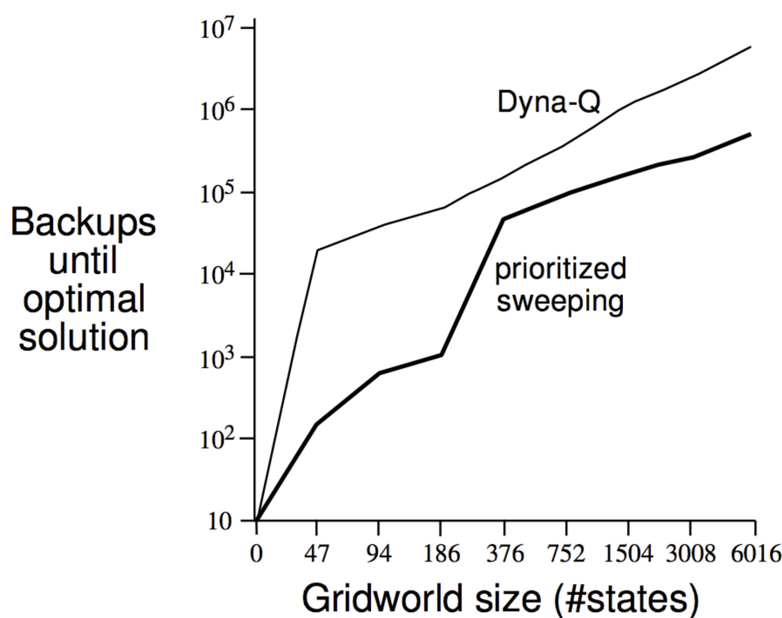
Work backwards from states whose values have just changed:

- 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

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:
$s,a \leftarrow first(PQueue)$
$s',r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
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

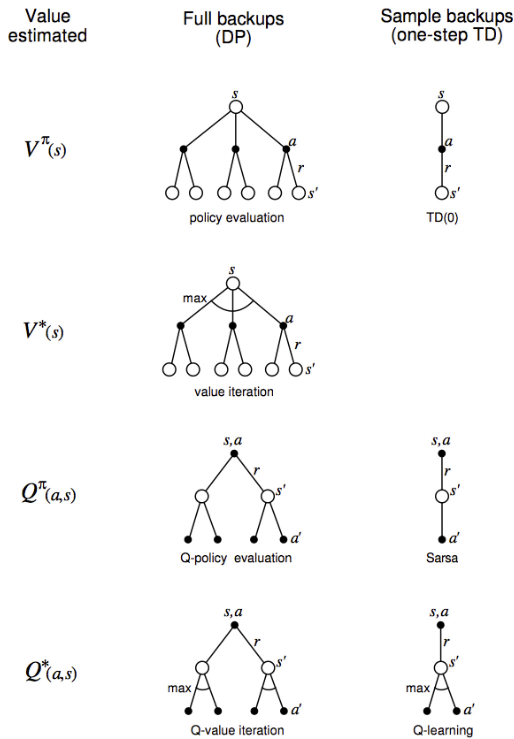


Data for a sequence of maze tasks which vary in the grid resolution.

Both use N=5 backups per environmental interaction.

From: Peng and Williams (1993)

Full and Sample (One-Step) Backups



Backups can be distinguished if they back-up **state** or **action values** and whether they estimate the value for the **optimal policy** or for an **arbitrary given policy**, or whether the backups are **full backups**, considering all possible events that might happen, or **sample backups**, considering a single sample of what might happen.

These three binary dimensions give rise to eight cases, seven of which correspond to specific algorithms.

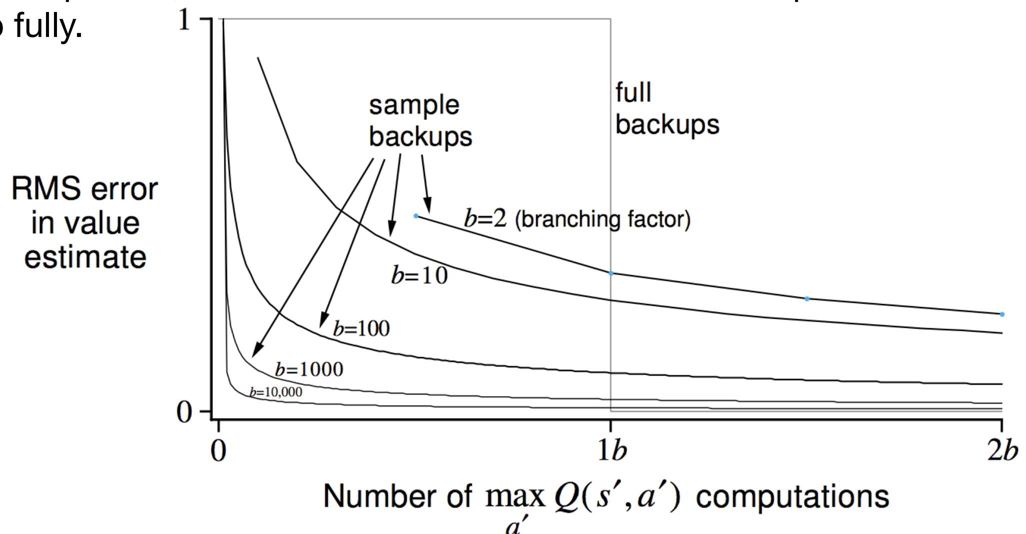
Full vs. Sample Backups

- If only one next state is possible, then the full and sample backups are identical
- The full backup is an exact computation, resulting in a new Q(s,a) whose correctness is limited only by the correctness of the Q(s',a') at successor states
- Sample backups is are affected by sampling error
- Sample backup is cheaper computationally because it considers only one next state, not all possible next states.

If there is enough time to complete a full backup, then the resulting estimate is generally better than that of sample backups because of the absence of sampling error.
But if there is insufficient time to complete a full backup, then sample backups are always preferable.

Full vs. Sample Backups

For moderately large *b* the error of sample backups falls with a tiny fraction of one full backup. For these cases, many state-action pairs could have their values improved, in the same time that one state-action pair could be backed up fully.



b successor states, equally likely; initial error = 1;
The values at the next states are assumed correct, so the full backup reduces the error to zero upon its completion.

Trajectory Sampling

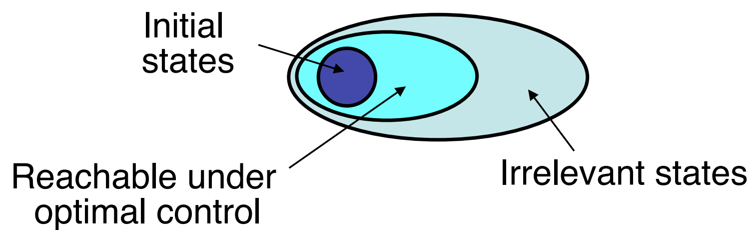
Dynamic programming:

- sweeps through the entire state space, backing up each state once per sweep.
- In many tasks the vast majority of the states are irrelevant because they are visited only under very poor policies or with very low probability.
- Exhaustive sweeps and the equal treatment of all states are not necessary properties of dynamic programming. Backups can be distributed any way one likes (to assure convergence, all states or state-action pairs must be visited in the limit an infinite number of times).

- Alternatively, sample from the state space according to some distribution, i) uniformly, or ii) on-policy distribution.

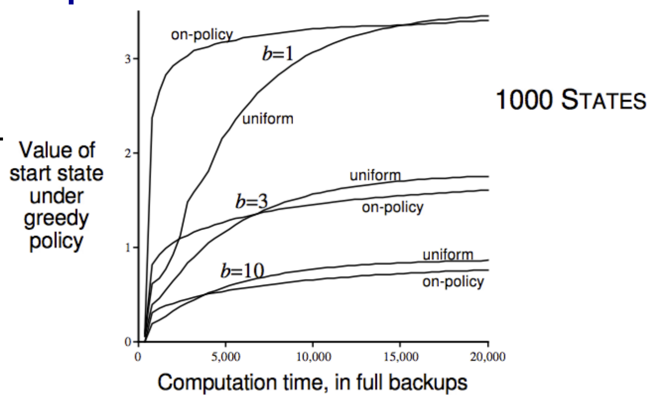
Trajectory Sampling

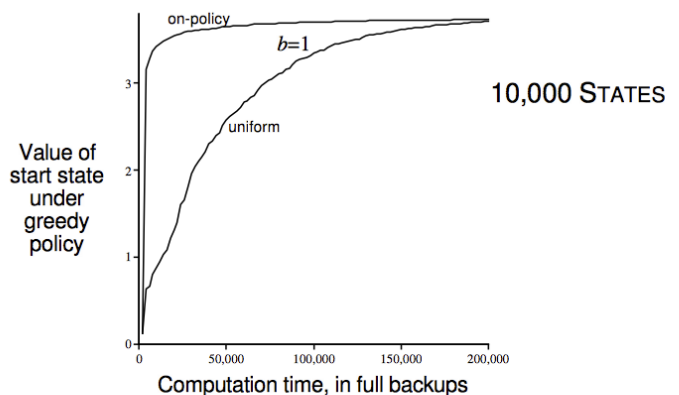
- Trajectory sampling: perform backups along simulated trajectories
- This samples from the on-policy distribution
- Advantages when function approximation is used
- Focusing of computation: can cause vast uninteresting parts of the state space to be (usefully) ignored:



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





Trajectory Sampling Experiment

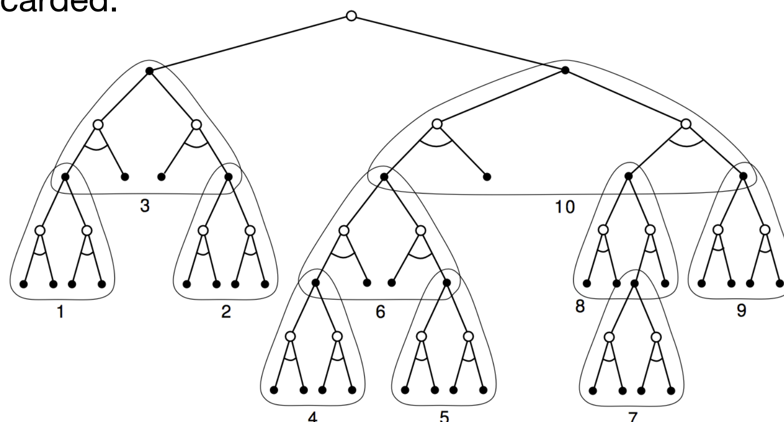
In the short term, sampling according to the on-policy distribution helps by focusing on states that are near descendants of the start state.

If there are many states and a small branching factor, this effect will be large and long-lasting.

In the long run, focusing on the on-policy distribution may hurt because the commonly occurring states all already have their correct values. Sampling them is useless, whereas sampling other states may actually perform some useful work. This presumably is why the exhaustive, unfocused approach does better in the long run, at least for small problems.

Heuristic Search

- Used for action selection, not for changing a value function (=heuristic evaluation function)
- The approximate value function is applied to the leaf nodes and then backed up toward the current state.
- The backing up within the search tree is just the same as in the max-backups discussed earlier.
- Once the backed-up values of these nodes are computed, the best of them is chosen as the current action, and then all backed-up values are discarded.



Heuristic Search

- No effort is made to save the backed-up values by changing the approximate value function. The value function is generally designed by people and never changed as a result of search.
- Searching deeper than one step is to obtain better action selections. If one has a perfect model and an imperfect action-value function, then in fact deeper search will usually yield better policies
- Also suggests ways to select states to backup: smart focusing:
  - The performance improvement observed with deeper searches is not due to the use of multistep backups as such. Instead, it is due to the focus and concentration of backups on states and actions immediately downstream from the current state.

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