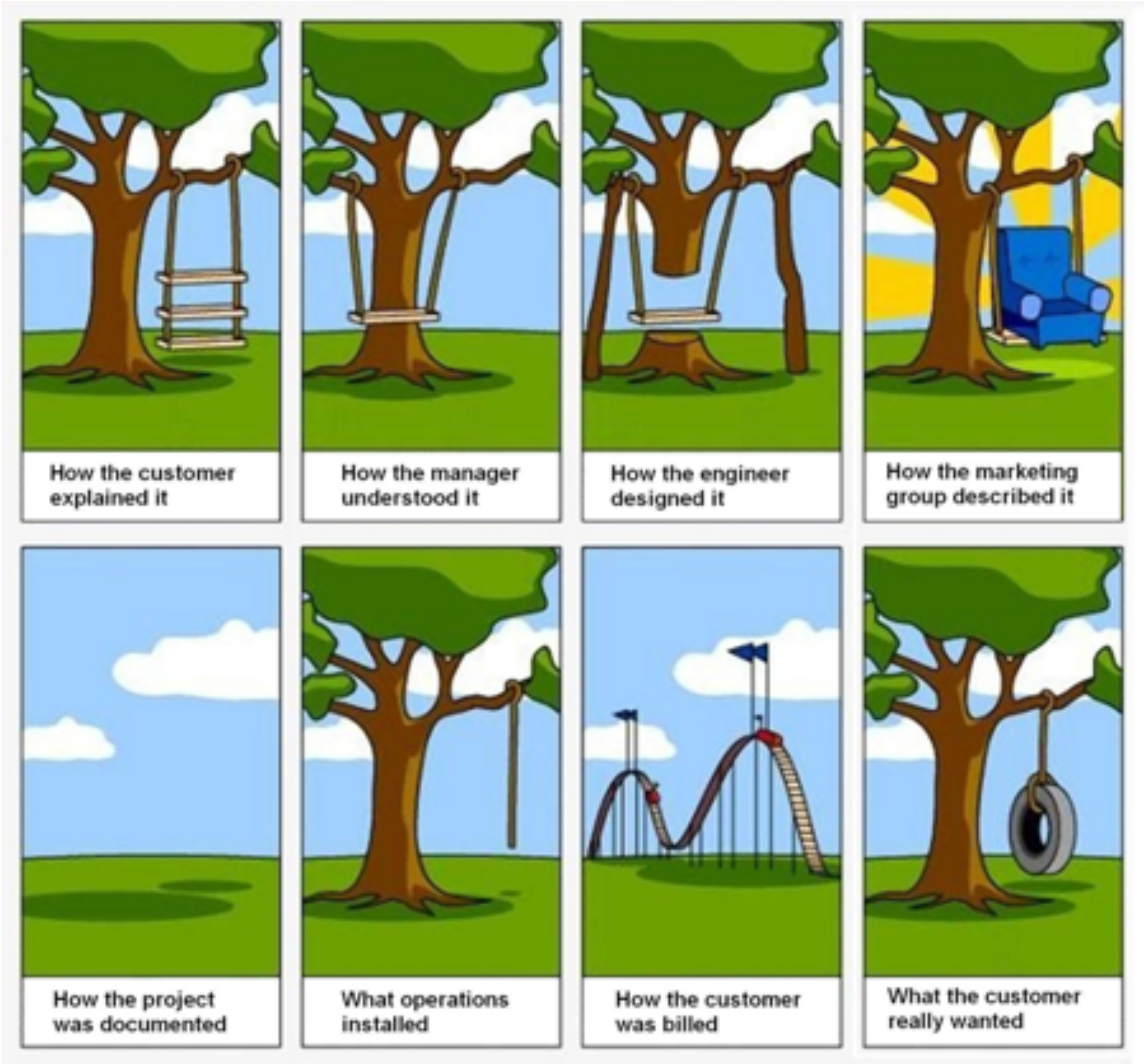
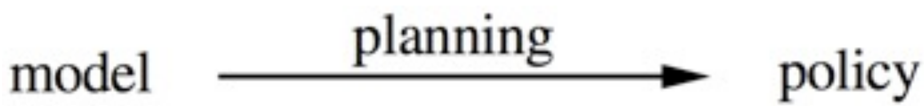
Efficient Planning

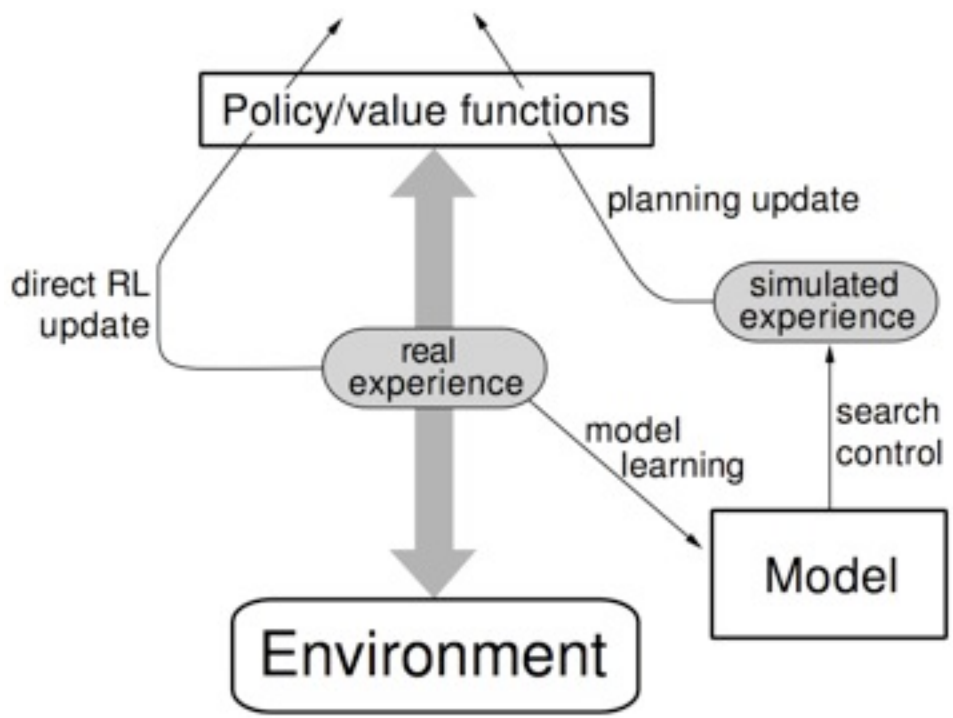


Tuesday class summary:

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



- Dyna framework:



Questions during class

- "Why use simulated experience? Can't you directly compute solution based on model?"
- "Wouldn't it be better to plan backwards from goal"

How to Achieve Efficient Planning?

- What type of backup is better?
  - Sample vs. full backups
  - Incremental vs. less incremental backups
- How to order the backups?

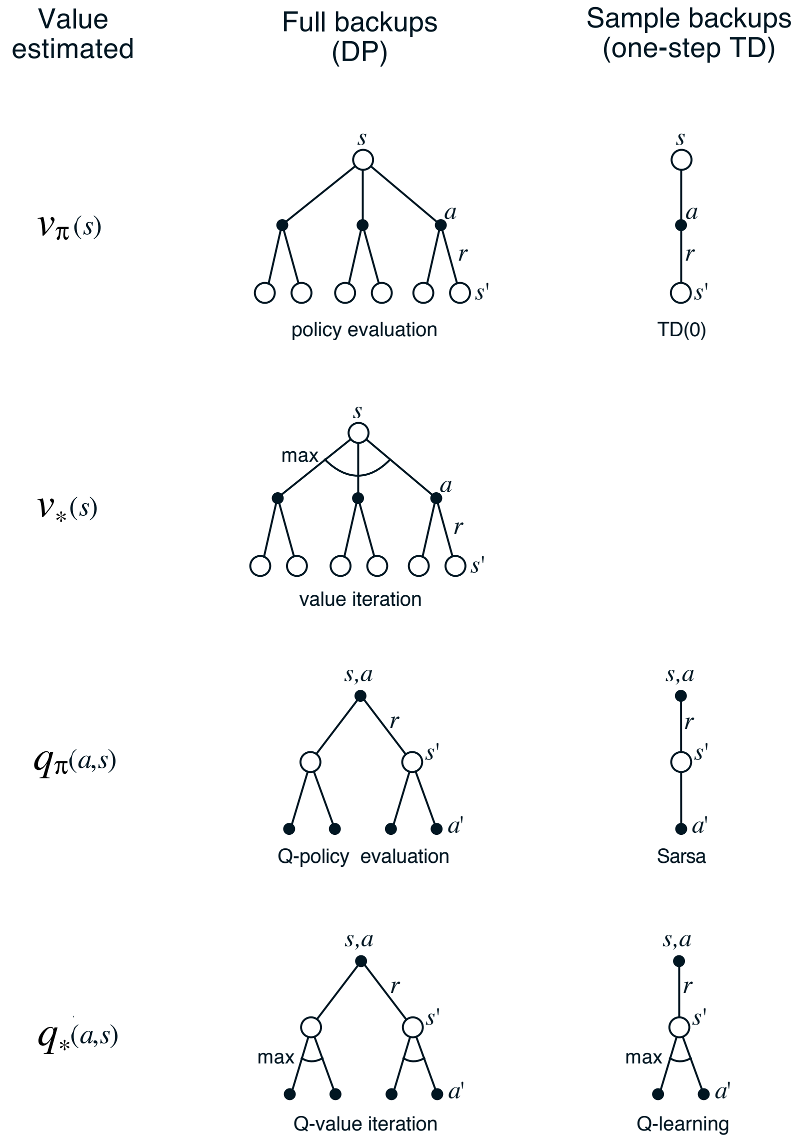
What is Efficient Planning?

Planning algorithm A is more efficient than planning algorithm B if:

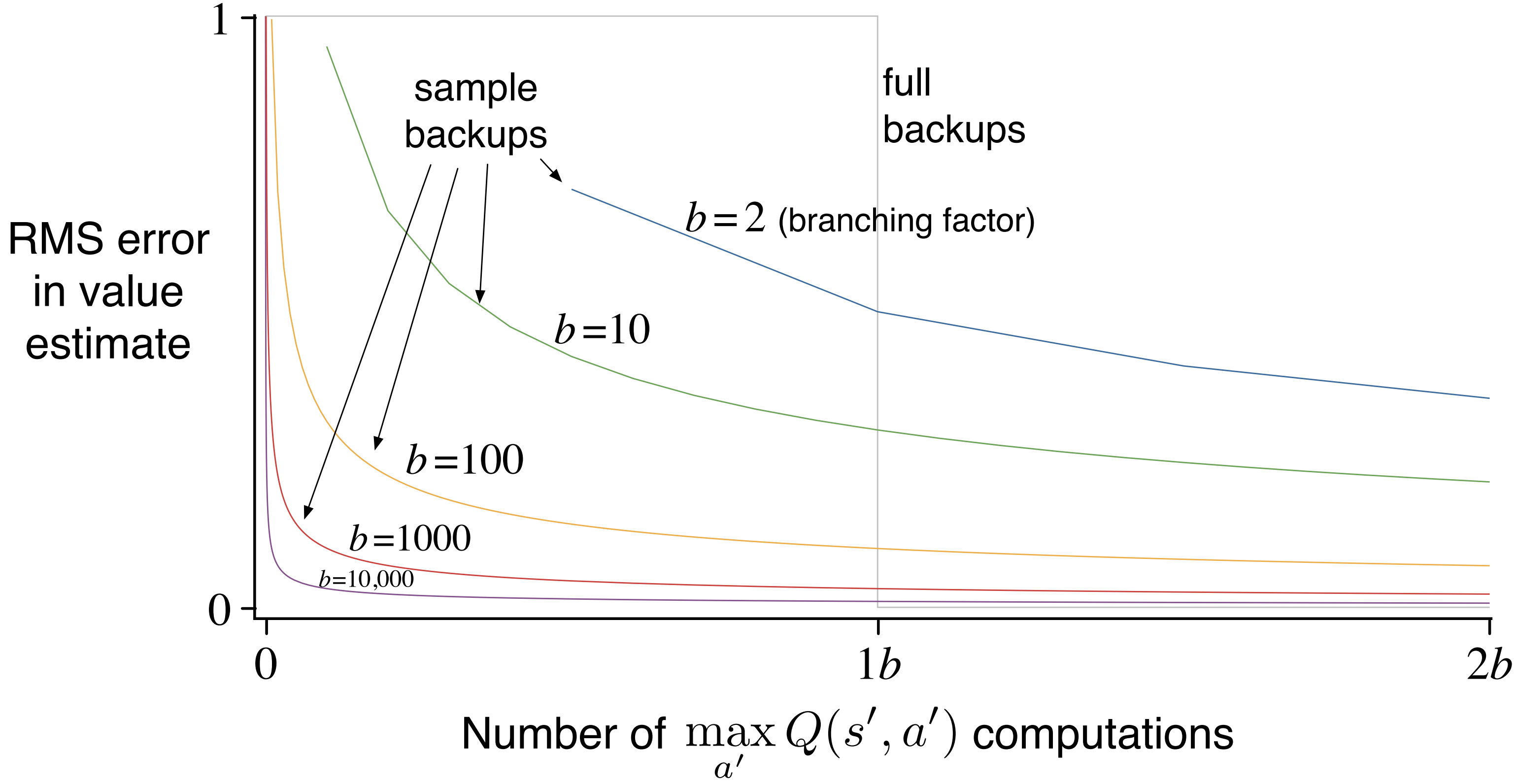
- it can compute the optimal policy (or value function) in less time.
- given the same amount of computation time, it improves the policy (or value function) more.

What backup type is best?

Full vs. Sample Backups



Full vs. Sample Backups



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

Small Backups

- Small backups are single-successor backups based on the model
- Small backups have the same computational complexity as sample backups
- Small backups have no sampling error
- Small backups require storage for ‘old’ values

Main Idea behind Small Backups

Consider estimate A that is constructed from a weighted sum estimates X_i .

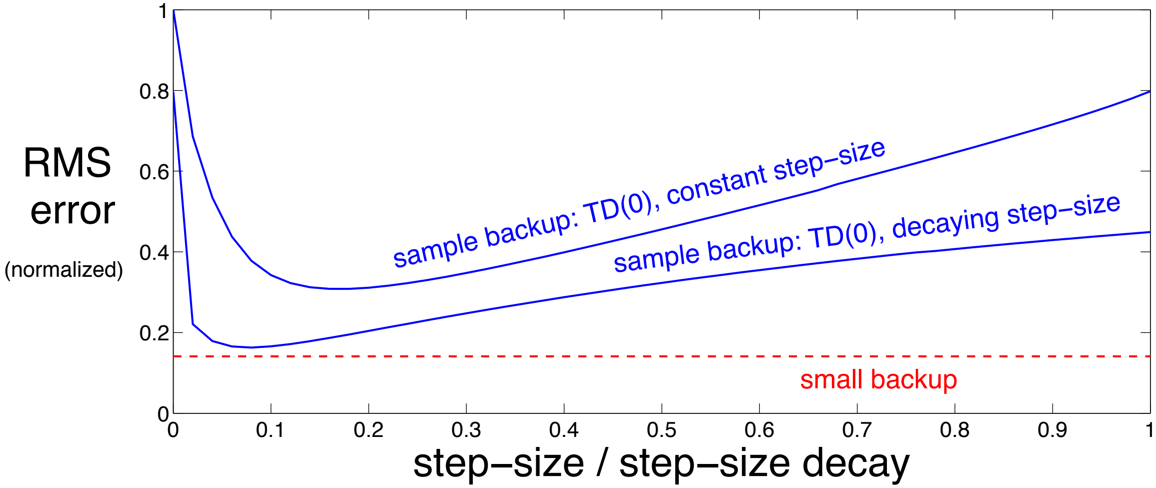
full backup: $$A \leftarrow \sum_i w_i X_i$$

What can we do if we know that only a single successor, X_j , changed value since the last backup?

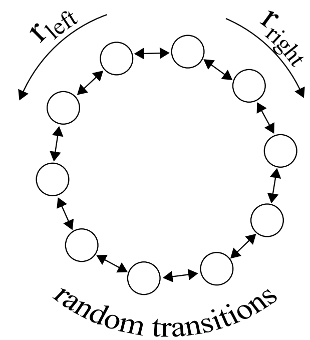
Let x_j be the old value of X_j , used to construct the current value of A. The value A can then be updated for a single successor by adding the difference between the new and the old value:

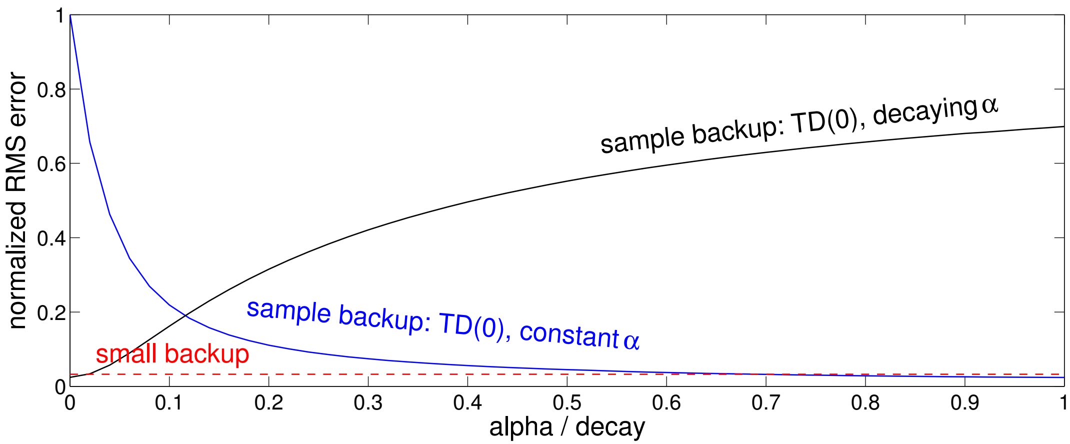
small backup: $$A \leftarrow A + w_j(X_j - x_j)$$

Small vs. Sample Backups



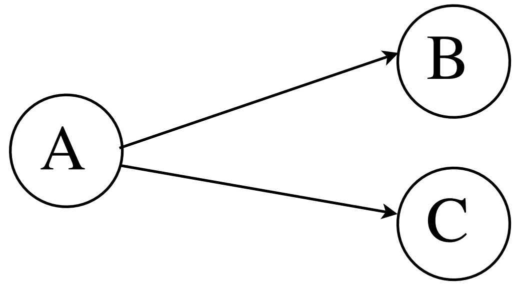
$r_{left} = +1$
$r_{right} = -1$



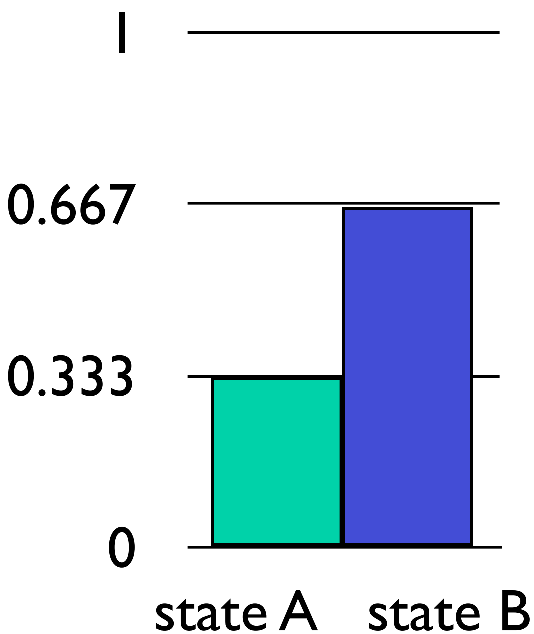


$r_{left} = +1$
$r_{right} = +1$

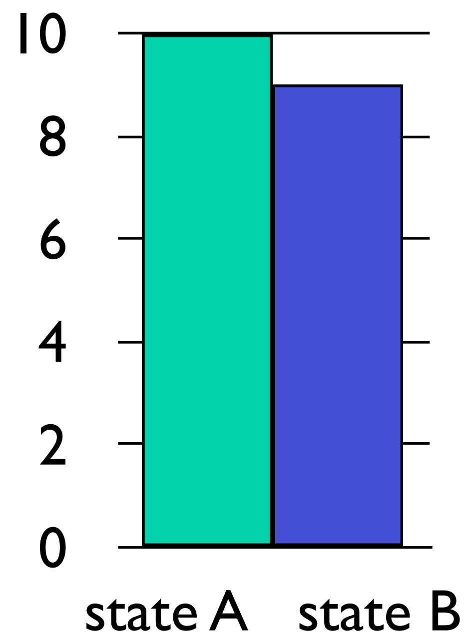
Small vs. Sample Backups



transition probability



state values



Backup Ordering

"I'm not sure officer, damn foreigners all look the same to me"

Backup Ordering

Do Forever:
1) Select a state $s \in \mathcal{S}$ according to some selection strategy H
2) Apply a full backup to s:

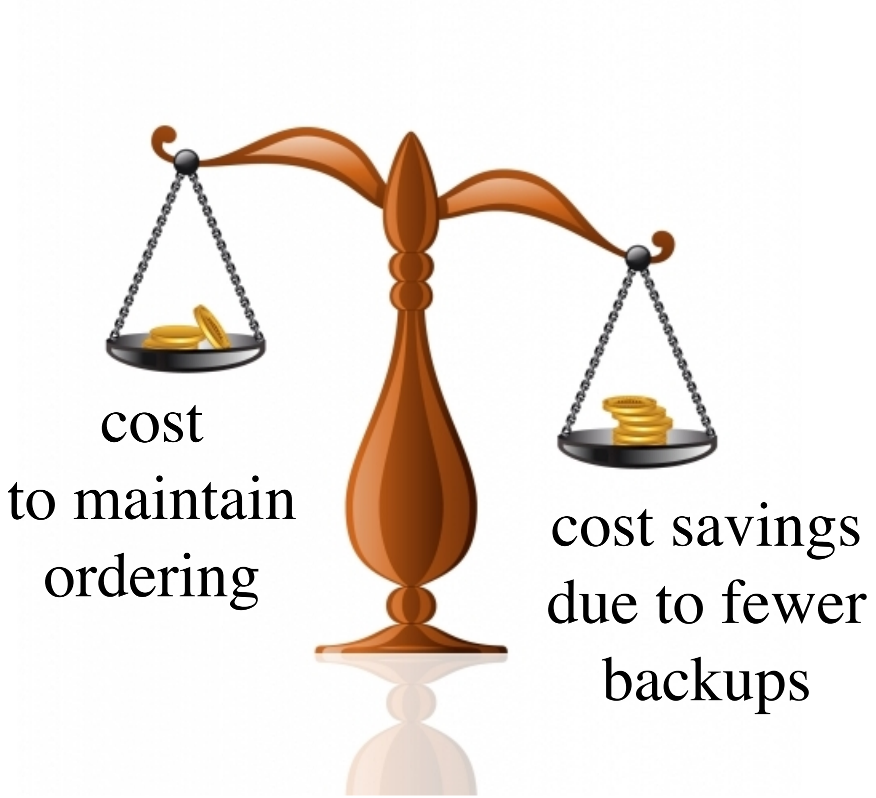
$$V(s) \leftarrow \max_a \left[\hat{r}(s,a) + \sum_{s'} p(s'|s,a) V(s') \right]$$

Asynchronous Value Iteration

- For every selection strategy H that selects each state infinitely often the values V converge to the optimal value function V_*
- The rate of convergence depends strongly on the selection strategy H

The Trade-Off

- For any effective ordering strategy the cost that is saved by having to perform less backups should out-weigh the cost of maintaining the ordering:



Prioritized Sweeping

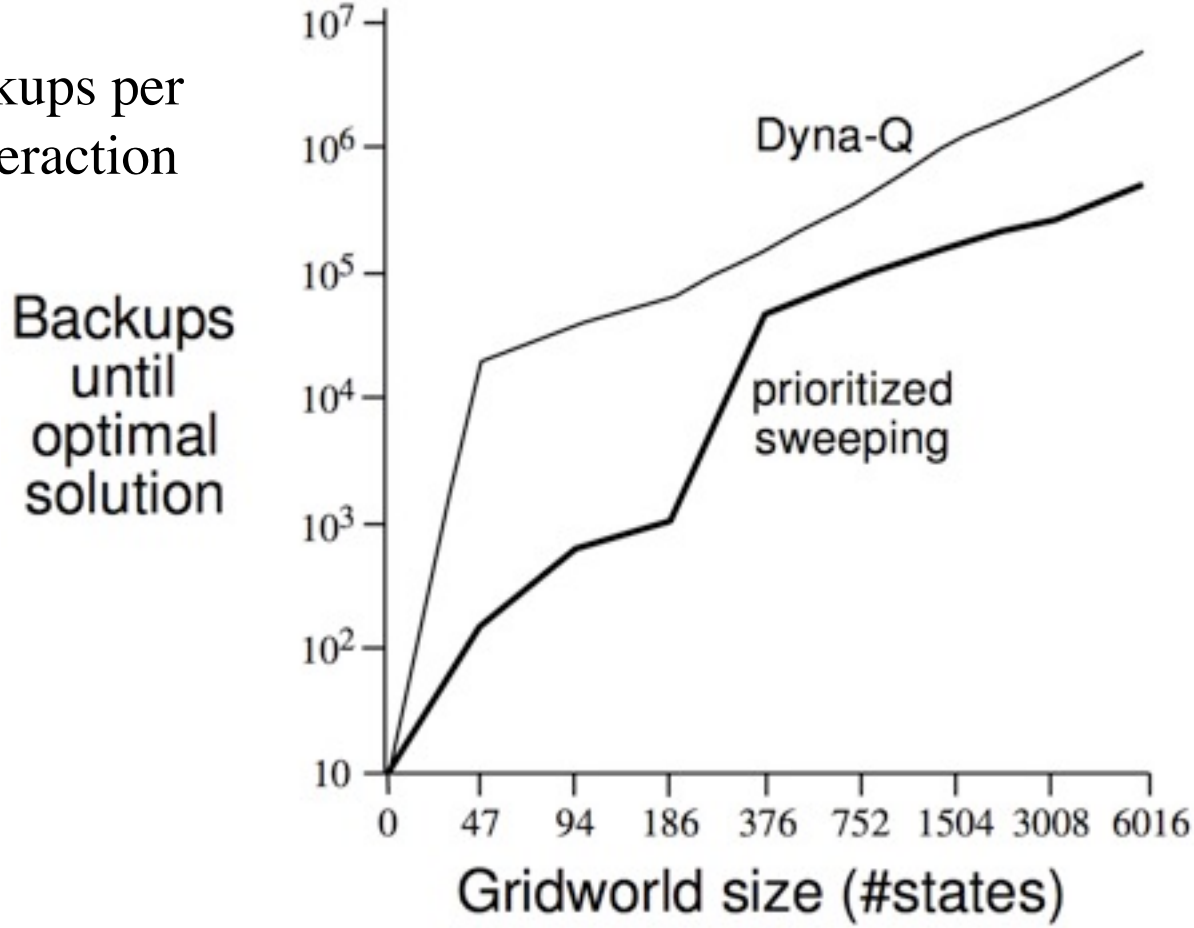
- Which states or state-action pairs should be generated during planning?
- Work backwards from states whose values have just changed:
  - Maintain a queue of state-action pairs whose values would change a lot if backed up, prioritized by the size of the change
  - When a new backup occurs, insert predecessors according to their priorities
  - Always perform backups from first in queue
- Moore & Atkeson 1993; Peng & Williams 1993
- improved by McMahan & Gordon 2005; Van Seijen 2013

Moore and Atekson’s Prioritized Sweeping

Published in 1993.

Prioritized Sweeping vs. Dyna-Q

Both use *n*=5 backups per environmental interaction



Bellman Error Ordering

- Bellman error is a measure for the difference between the current value and the value after a full backup:

$$BE(s) = \left| V(s) - \max_a \left[\hat{r}(s,a) + \sum_{s'} p(s'|s,a)V(s') \right] \right|$$

Bellman Error Ordering

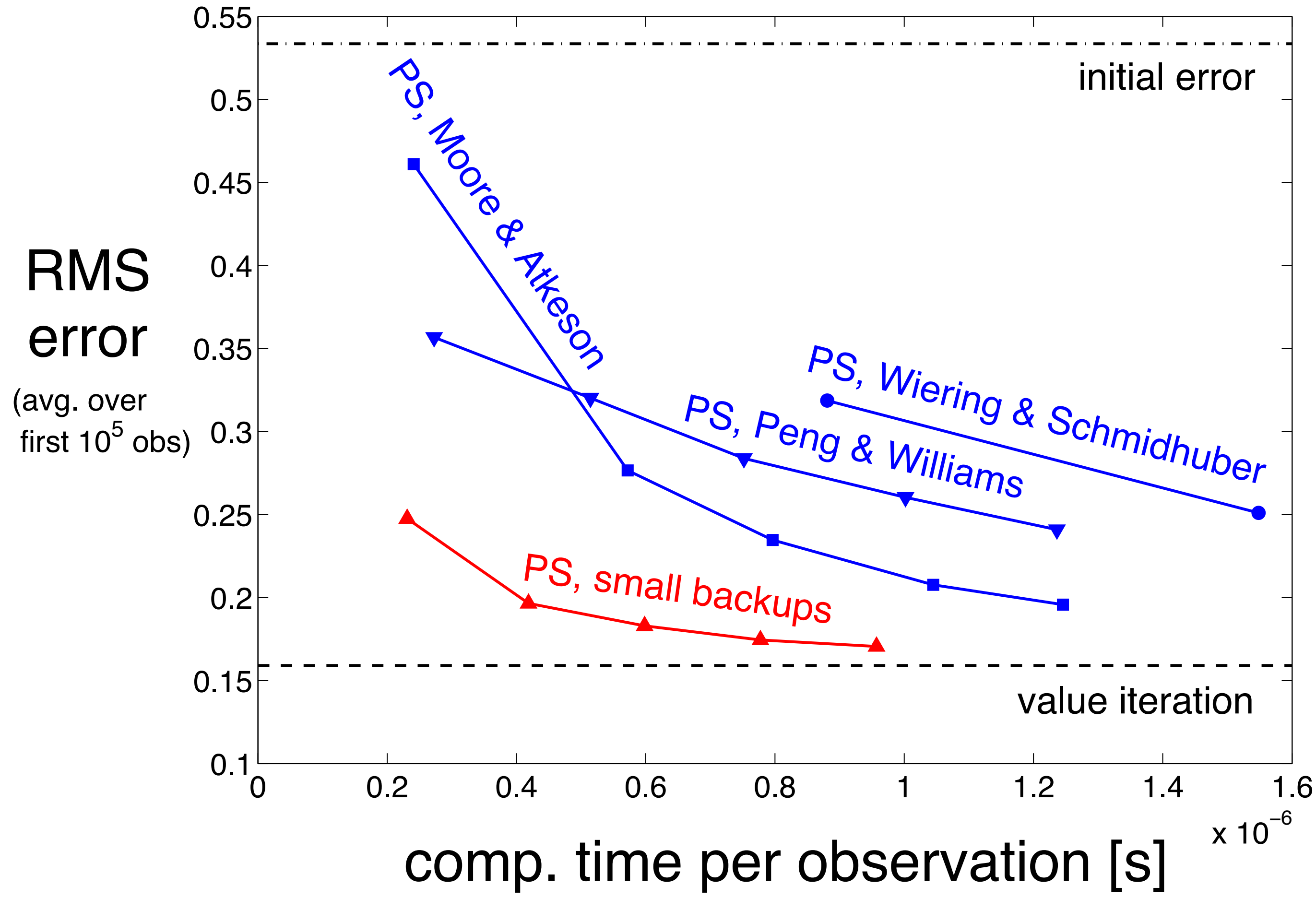
```
initialize V(s) arbitrarily for all s
compute BE(s) for all s
loop {until convergence}
   select state s' with worst Bellman error
   perform full backup of s'
   BE(s') ← 0
   for all predecessor states s̄ of s' do
      recompute BE(s̄)
   end for
end loop
```

To get positive trade-off:
comp. time Bellman error << comp time Full backup

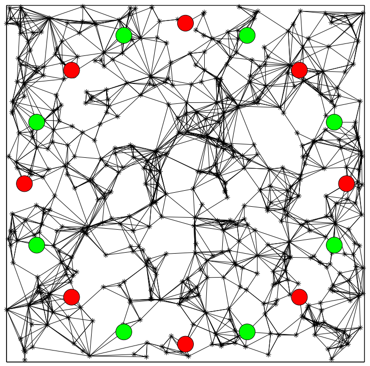
Prioritized Sweeping with Small Backups

initialize $V(s)$ arbitrarily for all s
initialize $U(s) = V(s)$ for all s
initialize $Q(s,a) = V(s)$ for all s, a
initialize $N_{sa}, N_{sa}^{s'}$ to 0 for all s, a, s'
loop {over episodes}
  initialize s
  repeat {for each step in the episode}
    select action a, based on $Q(s, \cdot)$
    take action a, observe r and s'
    $N_{sa} \leftarrow N_{sa} + 1; \quad N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
    $Q(s,a) \leftarrow \big[Q(s,a)(N_{sa} - 1) + r + \gamma V(s')\big]/N_{sa}$
    $V(s) \leftarrow \max_b Q(s,b)$
    $p \leftarrow |V(s) - U(s)|$
    if s is on queue, set its priority to p; otherwise, add it with priority p
    for a number of update cycles **do**
      remove top state $\bar{s}'$ from queue
      $\Delta U \leftarrow U(\bar{s}') - V(\bar{s}')$
      $V(\bar{s}') \leftarrow VU\bar{s}')$
      for all $(\bar{s}, \bar{a})$ pairs with $N_{\bar{s}\bar{a}}^{\bar{s}'} > 0$ **do**
        $Q(\bar{s}, \bar{a}) \leftarrow Q(\bar{s}, \bar{a}) + \gamma N_{\bar{s}\bar{a}}^{\bar{s}'}/N_{\bar{s}\bar{a}} \cdot \Delta U$
        $U(\bar{s}) \leftarrow \max_b Q(\bar{s}, b)$
        $p \leftarrow |V(\bar{s}) - U(\bar{s})|$
        if s is on queue, set its priority to p; otherwise, add it with priority p
      end for
    end for
    $s \leftarrow s'$
  until s is terminal
end loop

Empirical Comparison

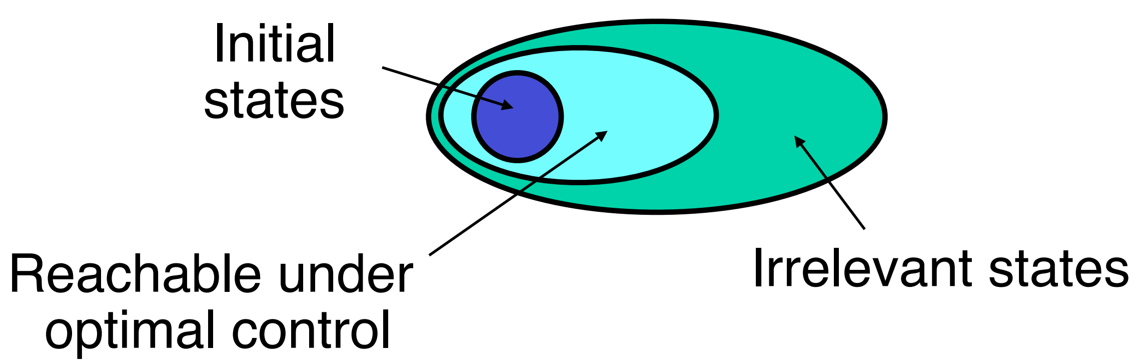


evaluation task:



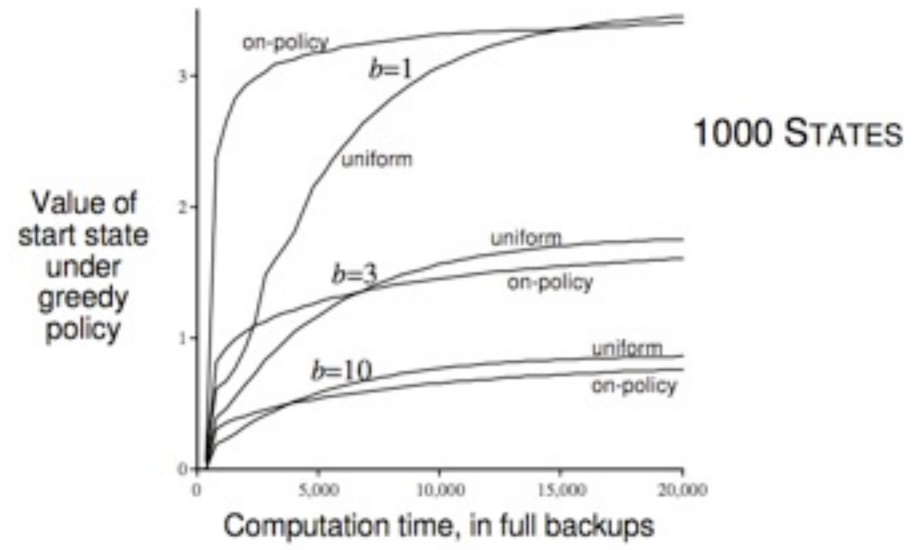
Trajectory Sampling

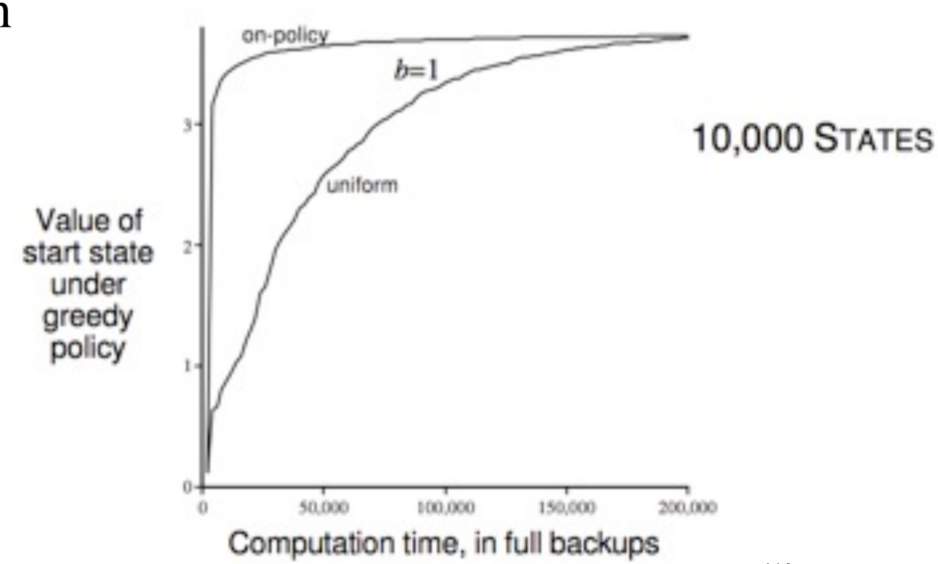
- Trajectory sampling: perform backups along simulated trajectories
- This samples from the on-policy distribution
- Advantages when function approximation is used (Chapter 8)
- Focusing of computation: can cause vast uninteresting parts of the state space to be (usefully) ignored:



Trajectory Sampling Experiment

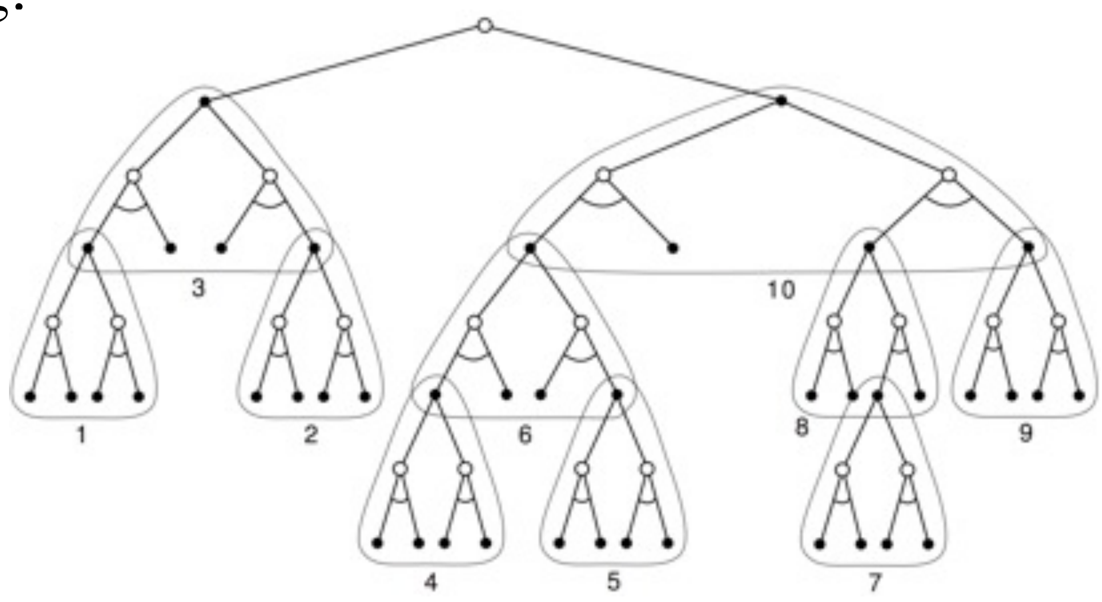
- one-step full tabular backups
- uniform: cycled through all state-action pairs
- on-policy: backed up along simulated trajectories
- 200 randomly generated undiscounted episodic tasks
- 2 actions for each state, each with *b* equally likely next states
- 0.1 prob of transition to terminal state
- expected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





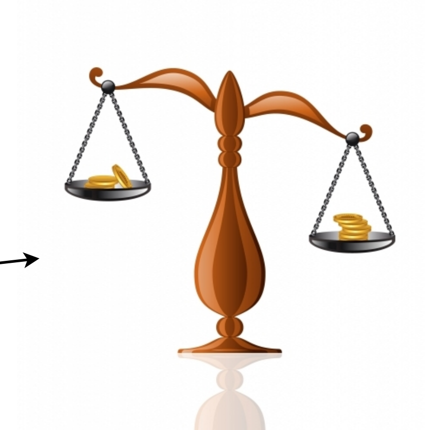
Heuristic Search

- 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

- Efficient planning is about trying to spend the available computation time in the most effective way.
- Backup types:
  - full/sample/small
- Backup Ordering
  - gain/loss trade-off
  - prioritized sweeping
  - prioritized sweeping with small backups: Bellman error ordering
  - trajectory sampling: backup along trajectories
  - heuristic search




That's all Folks!