Chapter 8 : Planning and Learning

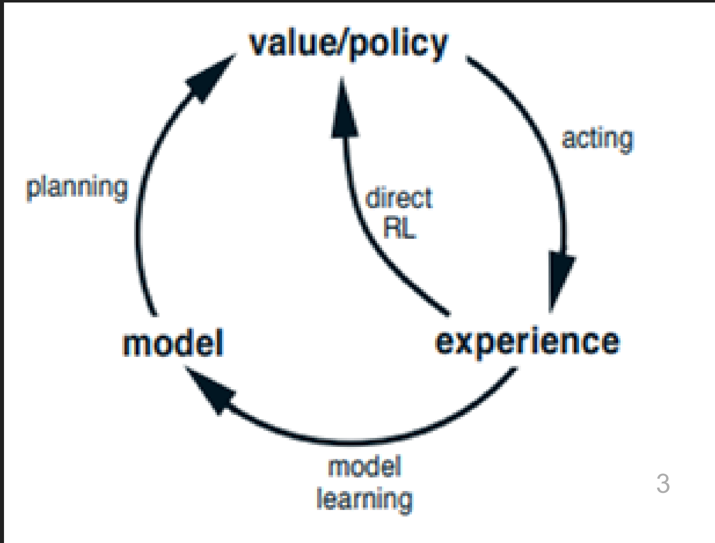
Key Concepts of Chapter 8

- Unifying Planning and Learning (model_based/model free).
- Prioritized Sweeping and Trajectory Sampling.
- Decision Time Planning (Monte Carlo Tree Search (Alpha Go)).

Planning and Learning

Planning : deriving experience from the model predictions.

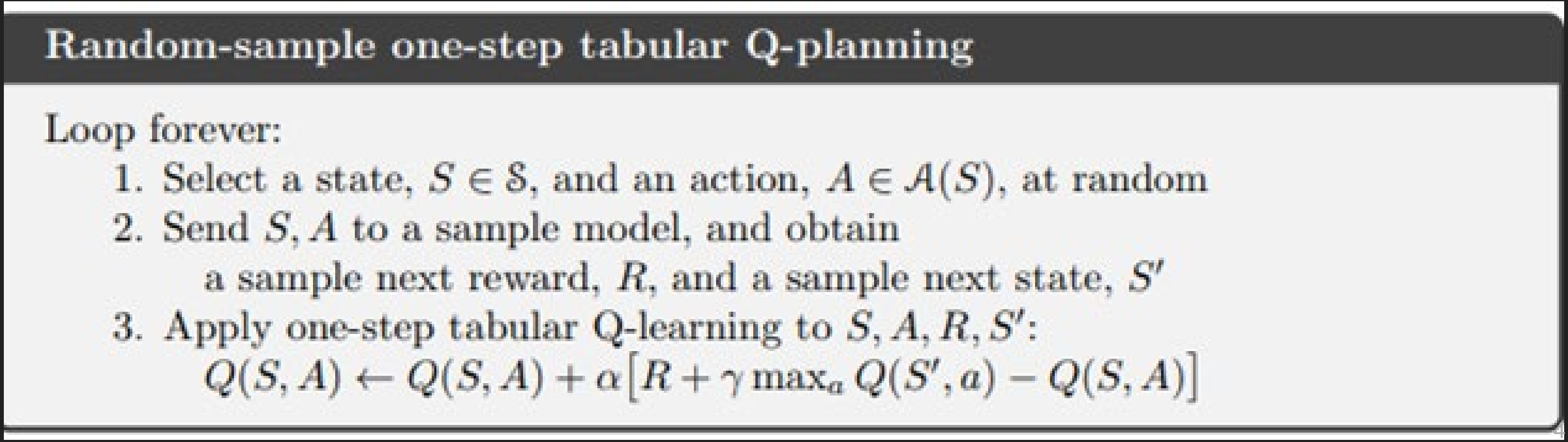
Learning : deriving experience from the real environment (trial and error).



Random-sample one-step tabular Q-planning

Sampling with a model

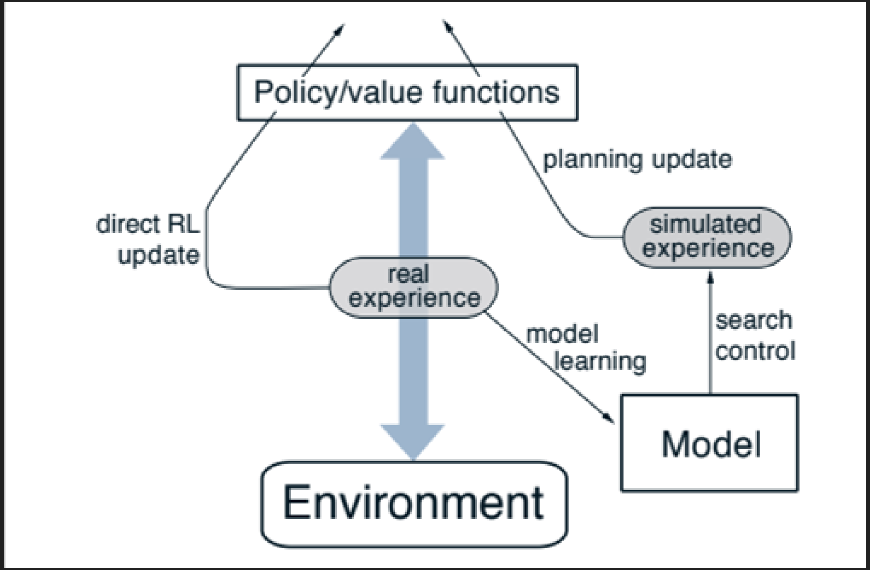
Updating with Q-learning algorithm



Random-sample one-step tabular Q-planning

```
Loop 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 reward, $R$, and a sample next state, $S'$
    3. Apply one-step tabular Q-learning to $S, A, R, S'$:
           $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
```

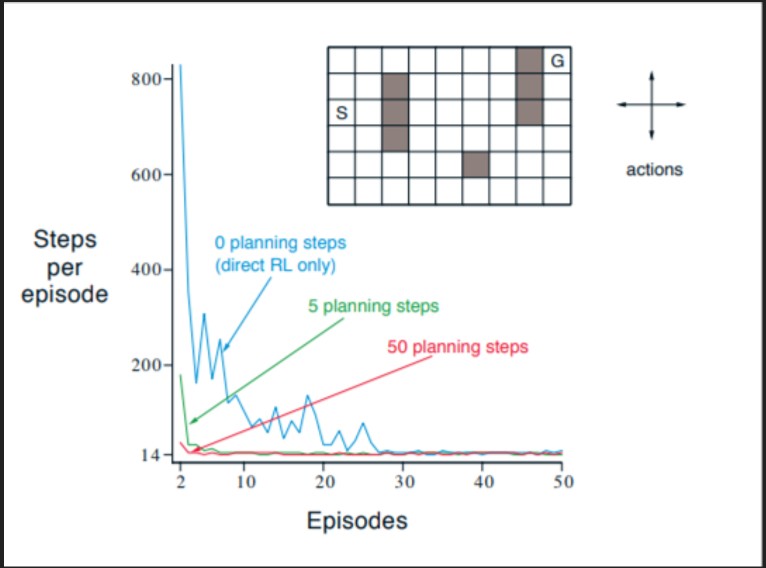
General Dyna Architecture



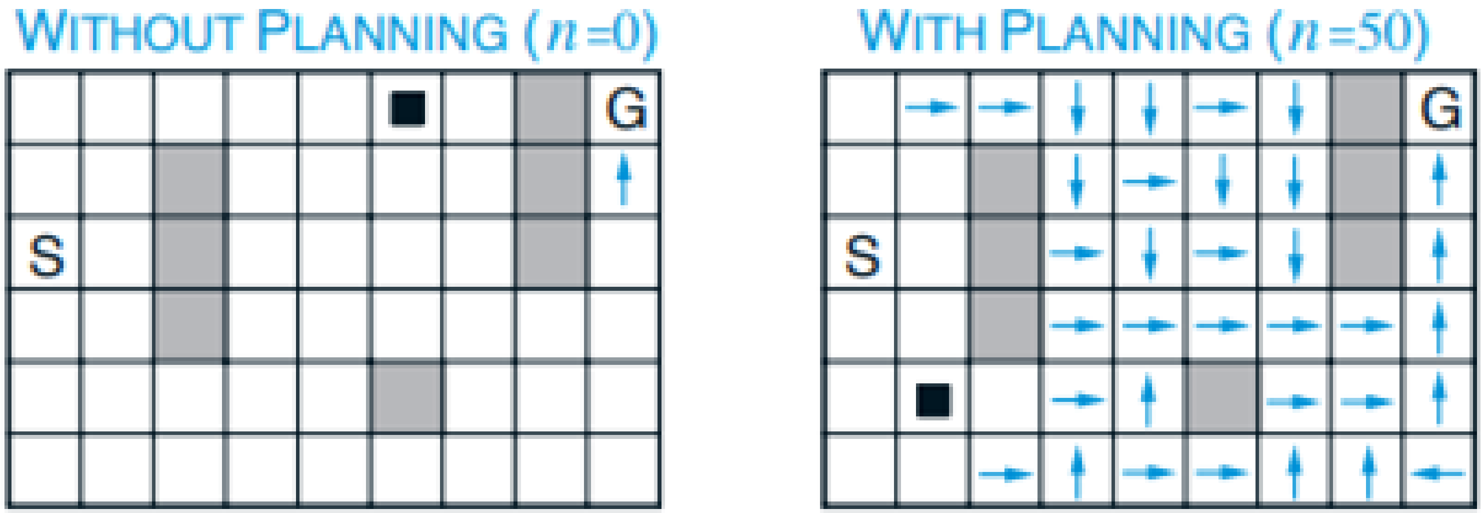
Impact of planning steps taken after real experience

0 planning steps : direct RL only => learning only from real world environment.

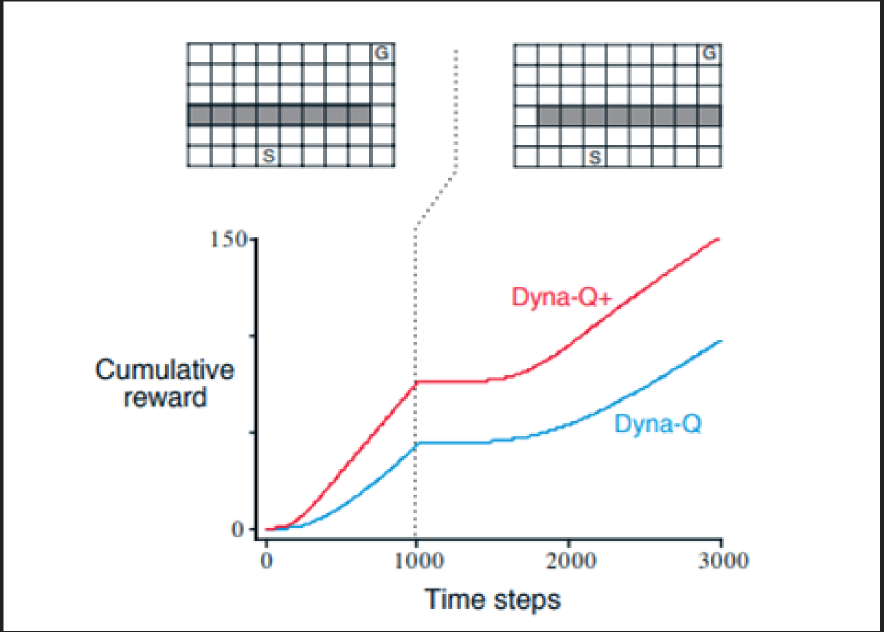
Reward of 0 on each transition to a non terminal state.

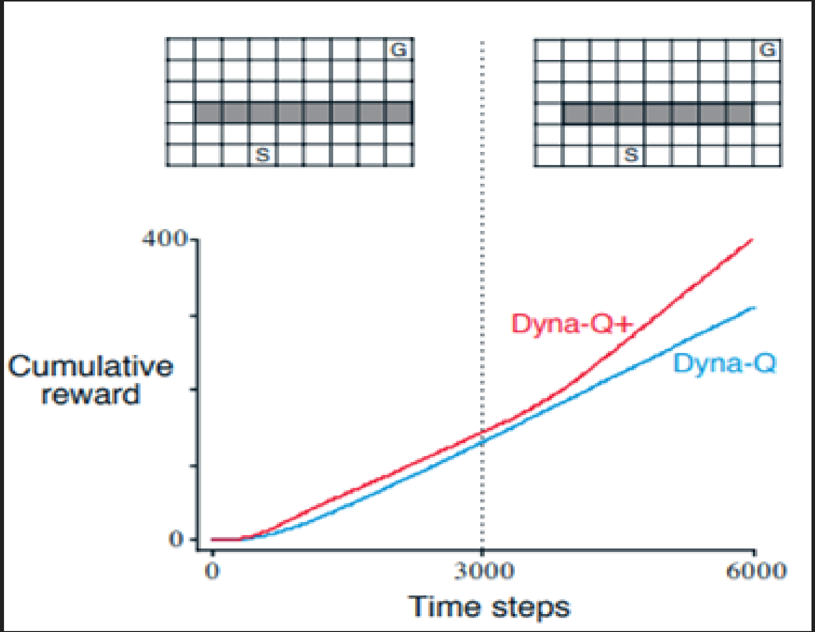


Impact of planning



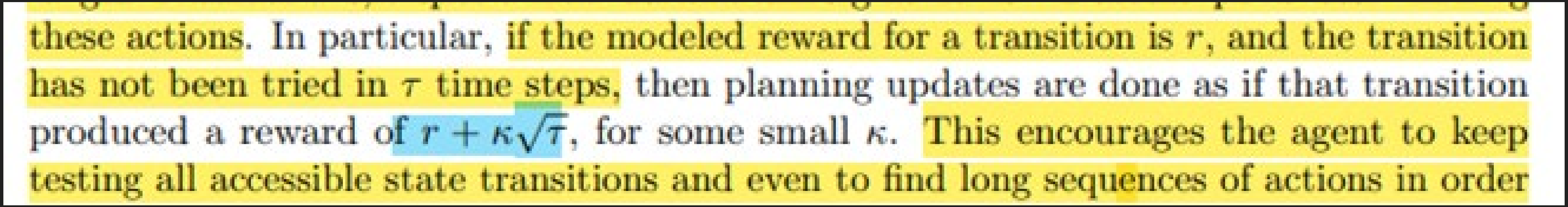
Planning in changing environments (When the Model is wrong)





Dyna vs. Dyna-Q

Dyna-Q adds a reward to states that haven't been experienced in the real world for a while (similar to UCB with e-greedy selection)


these actions. In particular, if the modeled reward for a transition is r, and the transition has not been tried in τ time steps, then planning updates are done as if that transition produced a reward of $r + \kappa\sqrt{\tau}$, for some small κ. This encourages the agent to keep testing all accessible state transitions and even to find long sequences of actions in order

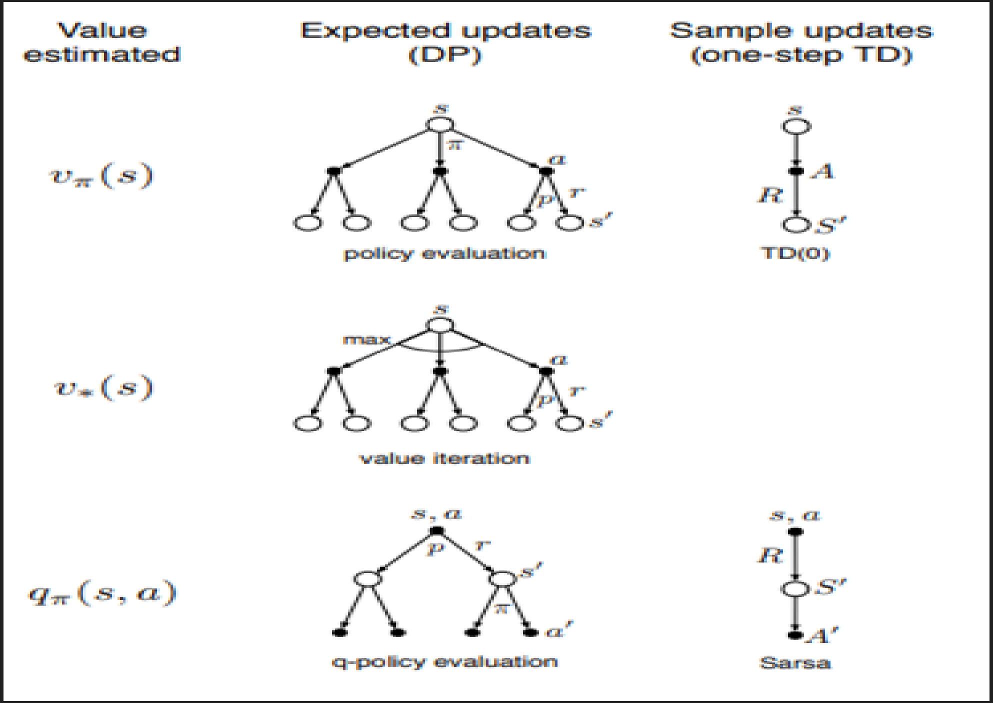
Expected vs Sample Updates

Branching factor b

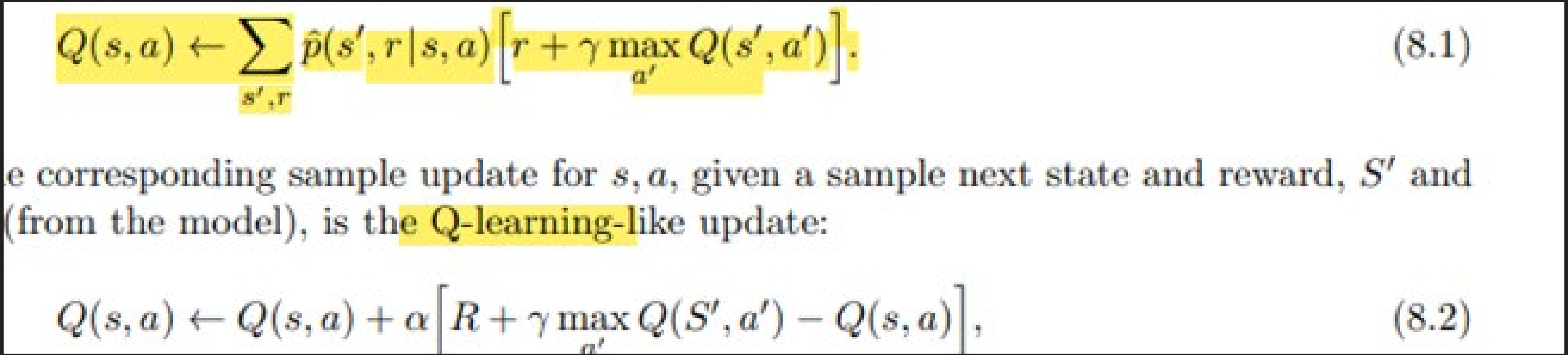
Expected updates certainly yield a better estimate.

Sampling error

Sampling advantage : good for large problems with many state action pairs / cheaper computationally.



Expected vs Sample Updates

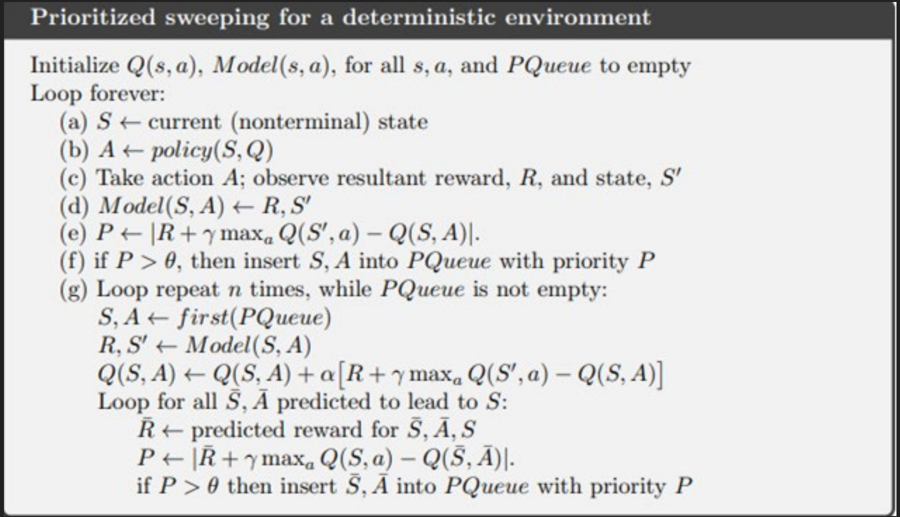


$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r|s,a)\Big[r + \gamma \max_{a'} Q(s',a')\Big]. \tag{8.1}$$

e corresponding sample update for s, a, given a sample next state and reward, S' and (from the model), is the Q-learning-like update:

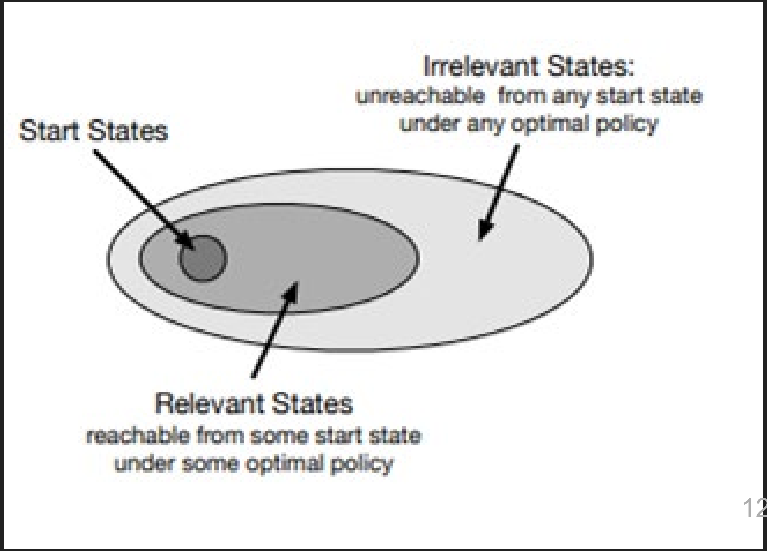
$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R + \gamma \max_{a'} Q(S',a') - Q(s,a)\Big], \tag{8.2}$$

Prioritized Sweeping

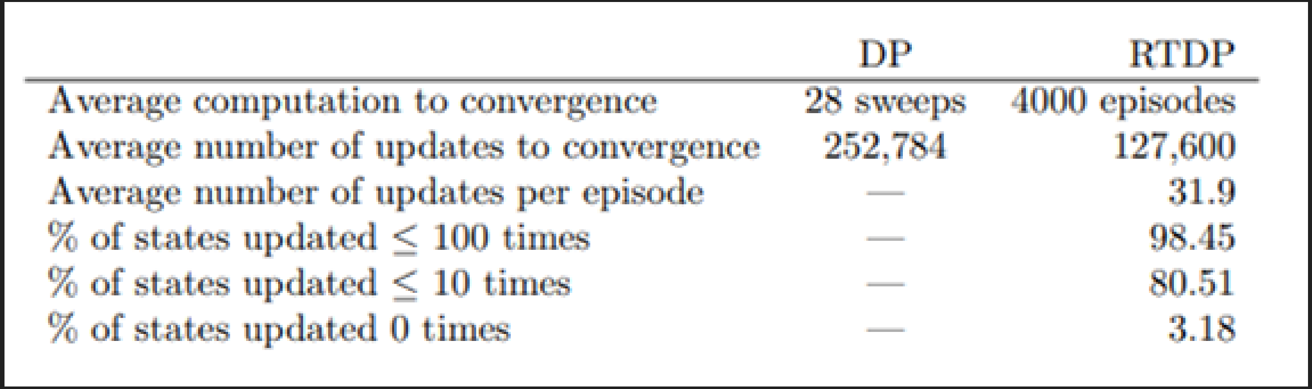


Prioritized sweeping for a deterministic environment

```
Initialize $Q(s,a)$, $Model(s,a)$, for all $s, a$, and $PQueue$ to empty
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow policy(S,Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Model(S,A) \leftarrow R, S'$
    (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) Loop repeat $n$ times, while $PQueue$ is not empty:
        $S, A \leftarrow first(PQueue)$
        $R, S' \leftarrow Model(S,A)$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        Loop for all $\bar{S}, \bar{A}$ predicted to lead to $S$:
            $\bar{R} \leftarrow$ predicted reward for $\bar{S}, \bar{A}, S$
            $P \leftarrow |\bar{R} + \gamma \max_a Q(S,a) - Q(\bar{S},\bar{A})|$.
            if $P > \theta$ then insert $\bar{S}, \bar{A}$ into $PQueue$ with priority $P$
```


Irrelevant States:
unreachable from any start state
under any optimal policy
Start States
Relevant States
reachable from some start state
under some optimal policy

Prioritized Sweeping : Racetrack Example

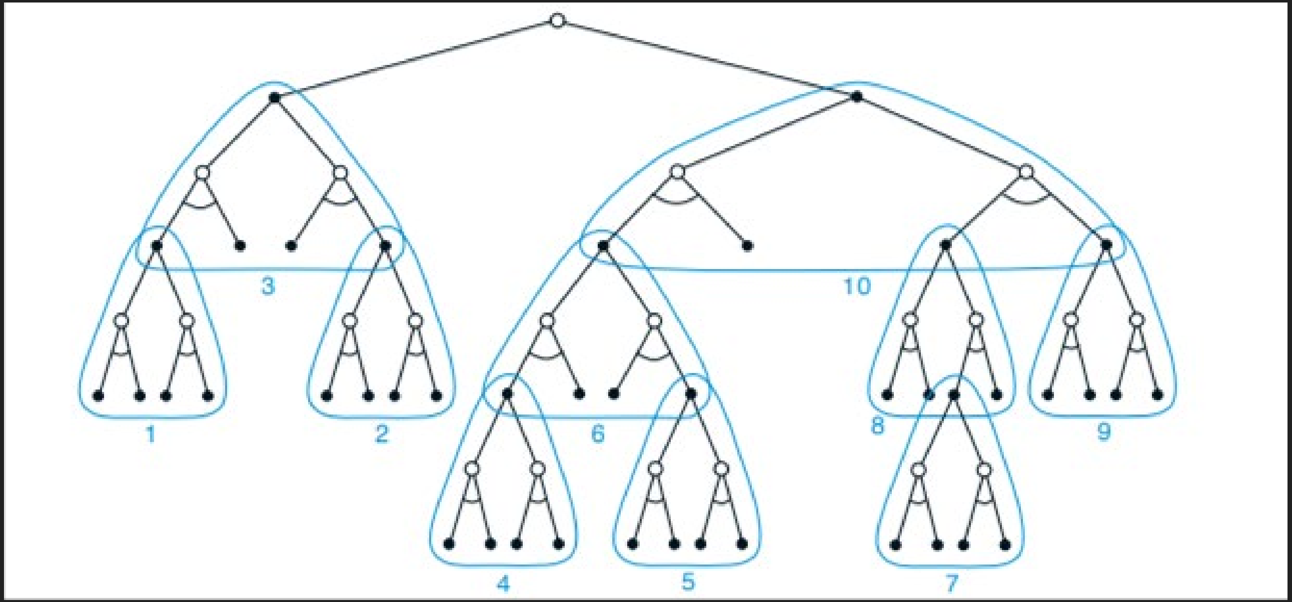


	DP	RTDP
Average computation to convergence	28 sweeps	4000 episodes
Average number of updates to convergence	252,784	127,600
Average number of updates per episode	—	31.9
% of states updated $\leq$ 100 times	—	98.45
% of states updated $\leq$ 10 times	—	80.51
% of states updated 0 times	—	3.18

Decision Time Planning

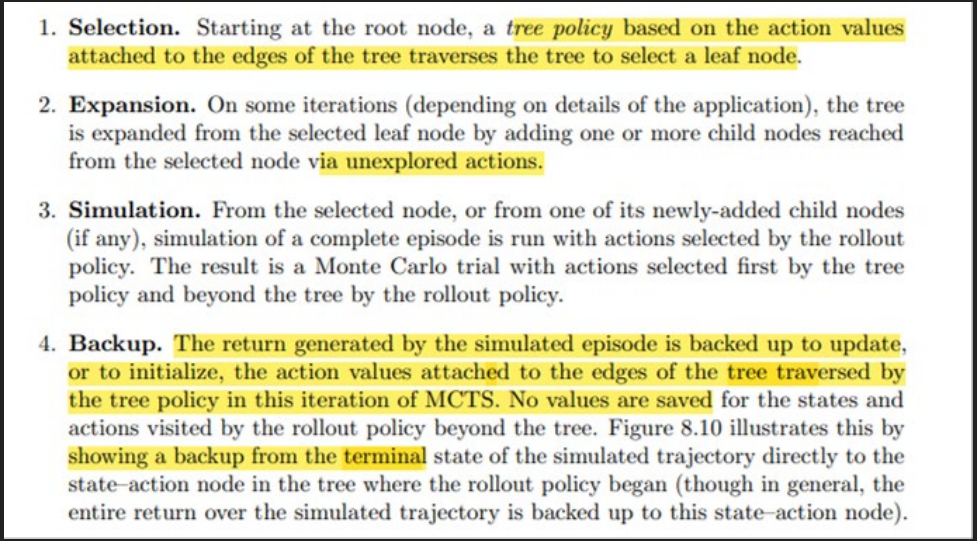
Heuristic Search

- The point of searching deeper than one step is to obtain better action selections

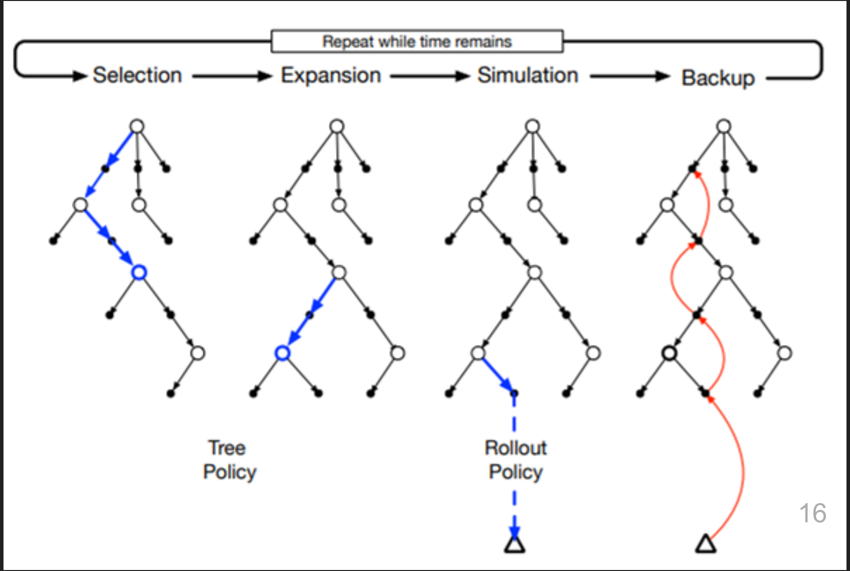


Monte Carlo Tree Search

MCTS is a rollout algorithm as described above, but enhanced by the addition of a means for accumulating value estimates obtained from the Monte Carlo simulations in order to successively direct simulations toward more highly-rewarding trajectories.



1. **Selection.** Starting at the root node, a *tree policy* based on the action values attached to the edges of the tree traverses the tree to select a leaf node.
2. **Expansion.** On some iterations (depending on details of the application), the tree is expanded from the selected leaf node by adding one or more child nodes reached from the selected node via unexplored actions.
3. **Simulation.** From the selected node, or from one of its newly-added child nodes (if any), simulation of a complete episode is run with actions selected by the rollout policy. The result is a Monte Carlo trial with actions selected first by the tree policy and beyond the tree by the rollout policy.
4. **Backup.** The return generated by the simulated episode is backed up to update, or to initialize, the action values attached to the edges of the tree traversed by the tree policy in this iteration of MCTS. No values are saved for the states and actions visited by the rollout policy beyond the tree. Figure 8.10 illustrates this by showing a backup from the terminal state of the simulated trajectory directly to the state–action node in the tree where the rollout policy began (though in general, the entire return over the simulated trajectory is backed up to this state–action node).


Repeat while time remains
Selection
Expansion
Simulation
Backup
Tree Policy
Rollout Policy