Model-Based

Roberto Capobianco

Reinforcement Learning

SAPIENZA
UNIVERSITÀ DI ROMA

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA
unibs.it

Model Based RL

- Model-based RL: emphasizes planning
  - Model is not given, but learned
- Model-free RL: emphasizes learning
  - Model: anything that an agent can use to predict how environment will respond to actions
  - Distribution models: describe all possibilities and their probabilities
  - Sample models: produce just one possibility sampled according to probabilities

- Different but also very similar
  - Both rely on value function
  - Both use lookahead to future events
  - Both use backed-up values

Model

- Models can be used to mimic or simulate experience

- Sample models:
  - produce a possible transition, given a starting state and action
  - could produce an entire episode, given starting state and policy
- Distribution model
  - generates all possible transitions weighted by probabilities
  - could generate all possible episodes and their probabilities

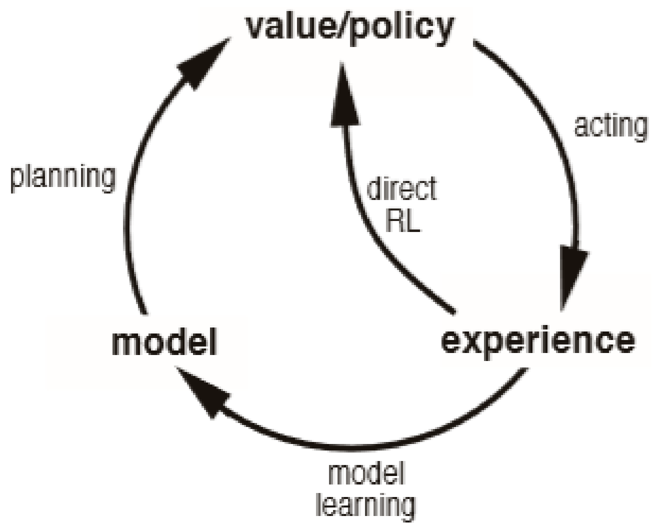
- In any case, model is used to simulate environment and produce simulated experience

Planning VS Learning

- Planning uses simulated experience generated by a model VS real experience generated by the environment
- Performance is assessed differently and experience can be generated with different flexibility
- Learning methods can often be substituted for update steps of planning methods

- Planning can be done online: while interacting with environment
- Interaction may change the model, interacting with planning
- Computation resource divided between model learning and decision making

- Experience can improve the model (model-learning)
- Experience can improve value function and policy using RL (direct RL)
- Policy and value function can be improved indirectly via model (indirect RL)



Q-Planning

- Indirect methods often make full use of limited experience
  - Achieve better policy with fewer environmental interactions

- Direct methods are often simpler and not affected by biases in model design

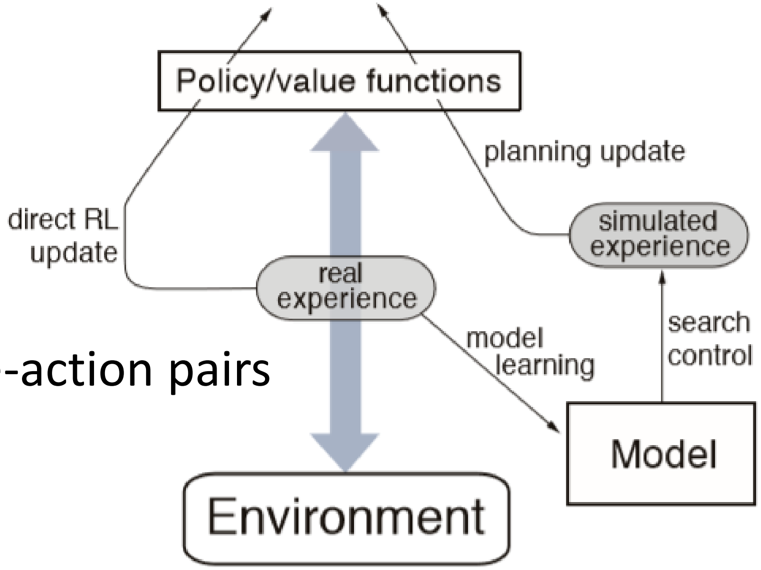
- Example: Q-Planning
  - converges to optimal policy for the model under same conditions of one-step Q-learning

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

Dyna-Q

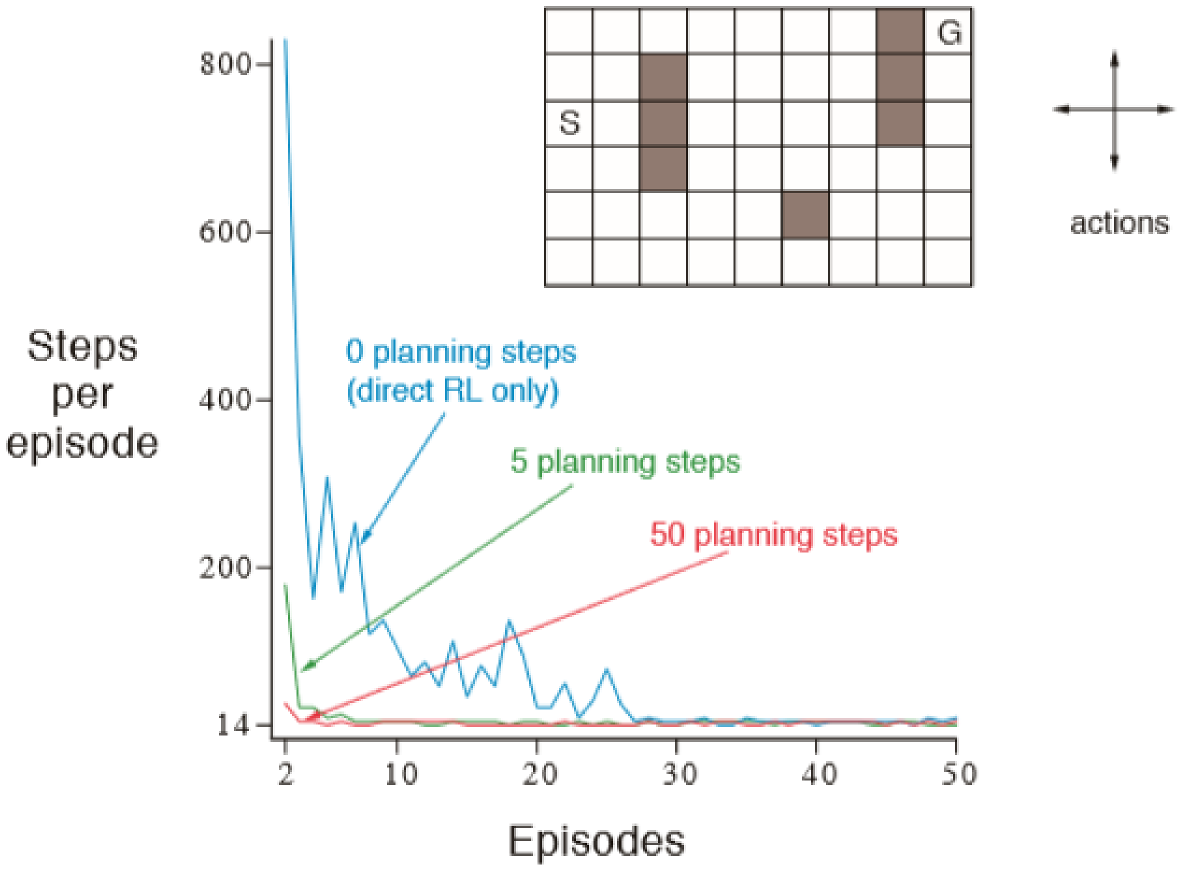
- Uses planning, acting, model-learning and direct RL
  - Planning: one-step tabular Q-planning
  - Direct RL: one-step tabular Q-learning
  - Model-learning: table based
    - Record transitions and assume they deterministically happen
    - Returns last observed next state and reward as prediction for state-action pairs
    - Queries can be done only against experienced state-action pairs



- Conceptually everything happens in parallel and simultaneously
- For implementation, we specify order

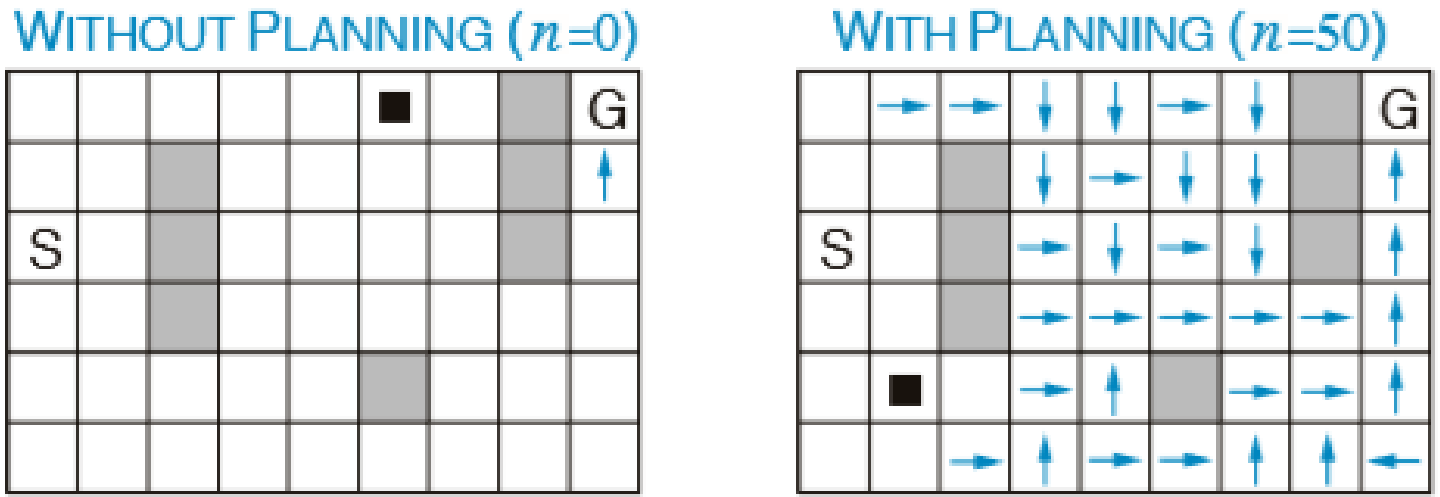
Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
  (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
  (f) Loop repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$



Dyna-Q Example

- After first episode planning can do its job



Model Inaccuracies

- In general models may be incorrect because environment is stochastic
  - Only limited number of samples is observed
  - Maybe we used also function approximation that generalized badly
  - Maybe environment changed

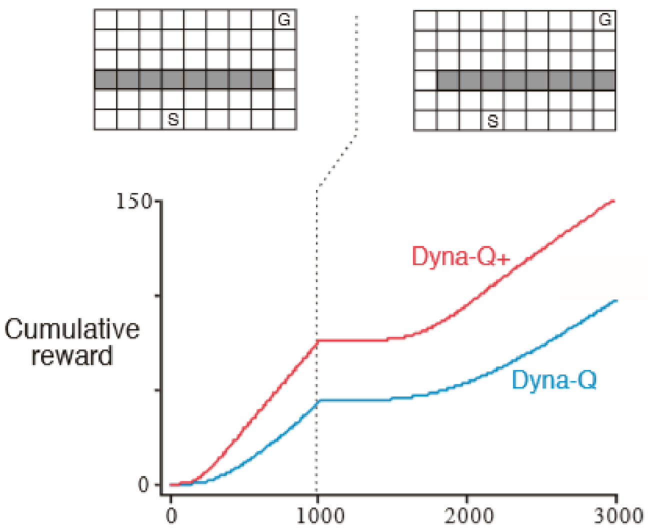
- If model is incorrect, planning generally computes suboptimal policy

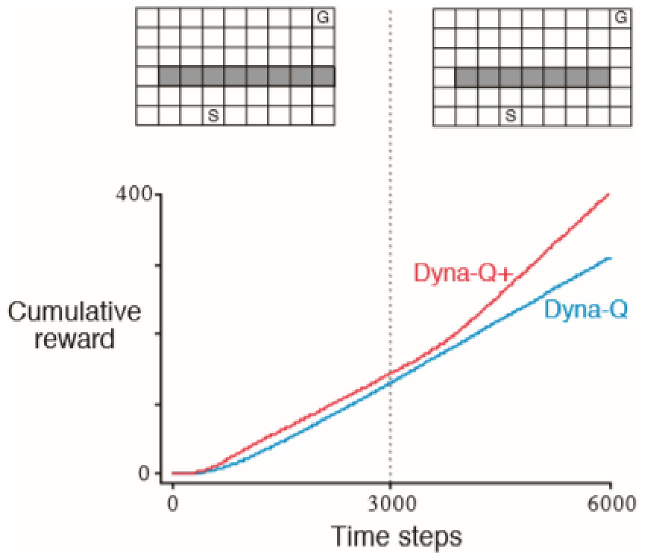
- Suboptimal policy may lead to discovery and correction of error
  - Happens if model is optimistic (predicts greater reward or better state)
  - Planned policy attempts to exploit these opportunities
  - Doing so it discovers that they do not exist
  - If environment gets better, policy does not reveal improvement

Model Exploration

- A different version of exploration Vs exploitation
  - Exploration: trying actions that improve the model
  - Exploitation: behaving optimally given current model
  - We want to improve model without degrading performance

- Dyna-Q+ uses one heuristic:
  - Keep track for each state-action pair of how many steps elapsed since last real visit
  - Longer time → greater chance it might have changed
  - Encourage behavior that tests untried actions



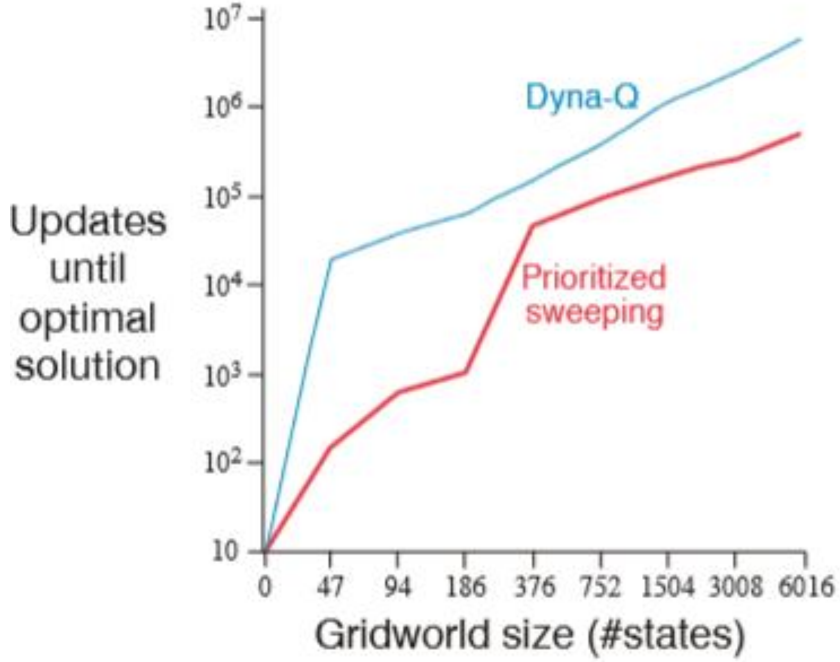


Focused Updates

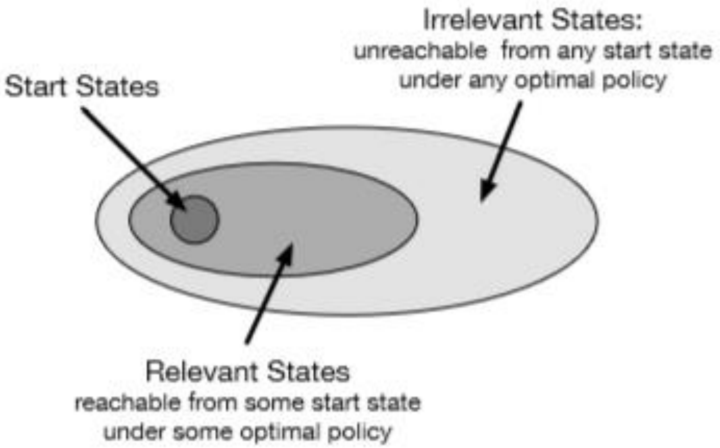
- In Dyna-Q state-action pairs are selected uniformly
  - We can do better by focusing
  - Avoid wasteful updates
  - Useful updates grow as planning progresses
- Intuition: we want to do backward updates for states whose value change
  - Value depends on lookahead in the future
  - Actions leading into states that have better value need to be updated
  - Their predecessors as well might have changed
- Idea: prioritize updates according to their urgency (**prioritized sweeping**)
  - Queue of state-action pairs whose value would change a lot
  - Prioritize by the size of change
  - Efficiently propagate effect on each of predecessor pairs
    - If effect is larger than a threshold, pair is also inserted in queue with priority

Prioritized Sweeping

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



- DP performs updates on each state
  - Problematic on large tasks: many states might be irrelevant
- Alternative:
  - Sample state or state-action space according to a distribution
    - Uniformly as Dyna-Q is bad (as for the full state space selection)
  - We can use on-policy distribution (observed following policy)
    - Easy to generate
    - Sample actions given by policy
    - Called **trajectory sampling**



	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

- Real-time DP: on-policy trajectory sampling version of value iteration (DP)
- Converges to optimal policies for discounted finite MDPs with exploring starts
- For certain types of problems it's guaranteed to find optimal policy on relevant states without visiting irrelevant states infinitely

Decision Time Planning

- Background planning (e.g., Dyna)
  - Not focused on current state
  - Gradually improve policy on the basis of simulated experience from model
  - Planning plays a part well before an action is selected

- Decision-time planning
  - Begin planning after encountering each new state
  - Evaluates action choices leading to different predicted states
  - Use simulated experience to select an action for the current state
  - Values and policy are updated specifically for current state

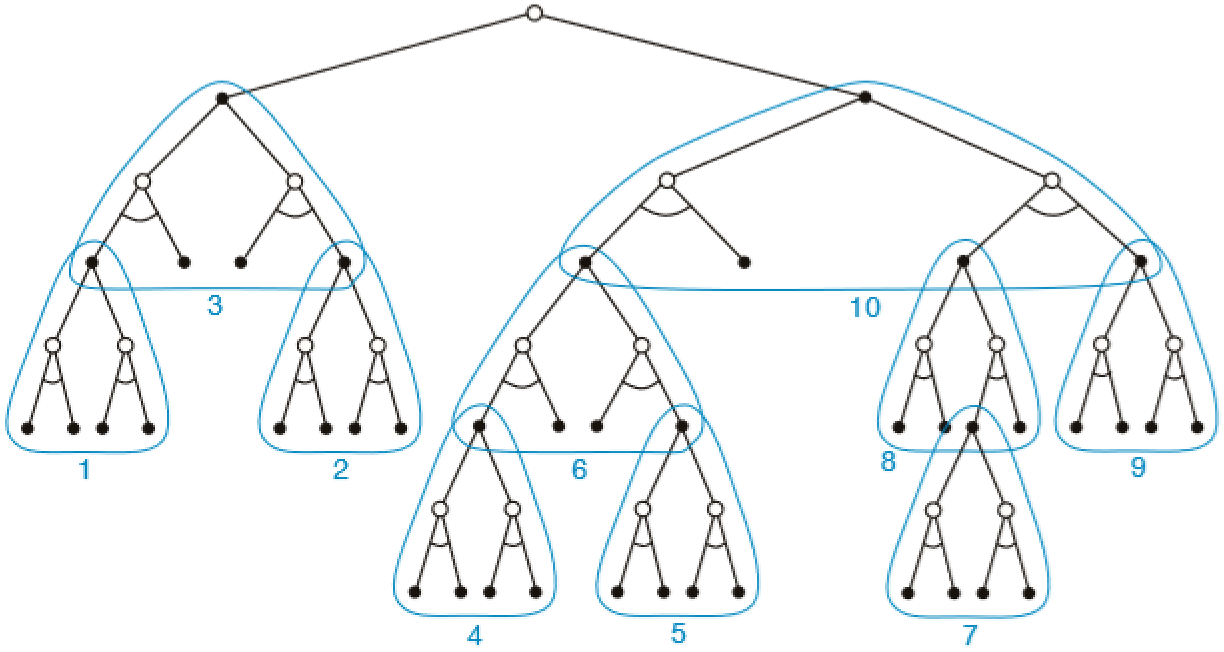
- Can be blended together

Heuristic Search

- Classical AI state-space planning
  - For each state met a large tree of possible continuations is evaluated
  - Approximate value function applied to leaf nodes and backed up
  - Best of values after back-up is chosen as current action
  - All backed-up values are (generally) discarded

- Value function generally hand-designed and never changed
  - This would be natural to do however
  - Greedy policies similar to one step heuristic search without saving back-up
- Focused on current state (memory and computational resources are focused)

- Limit case:
  - Use exactly methods of heuristic search to construct tree
  - Perform individual one-step updates from bottom up
  - In tabular case and with ordered updates: exactly like depth-first heuristic search
  - No multi-step update, but focused multiple one-step updates

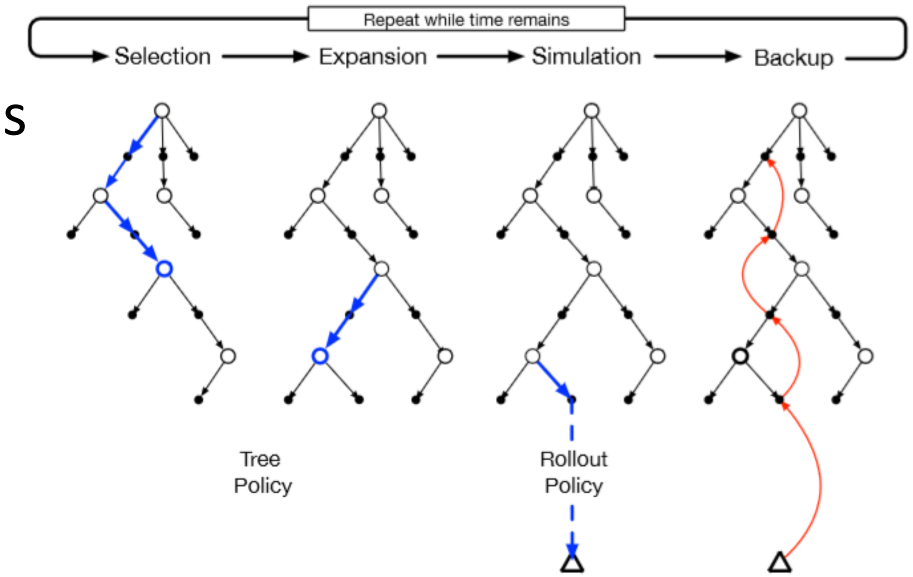


Roll-out Algorithms

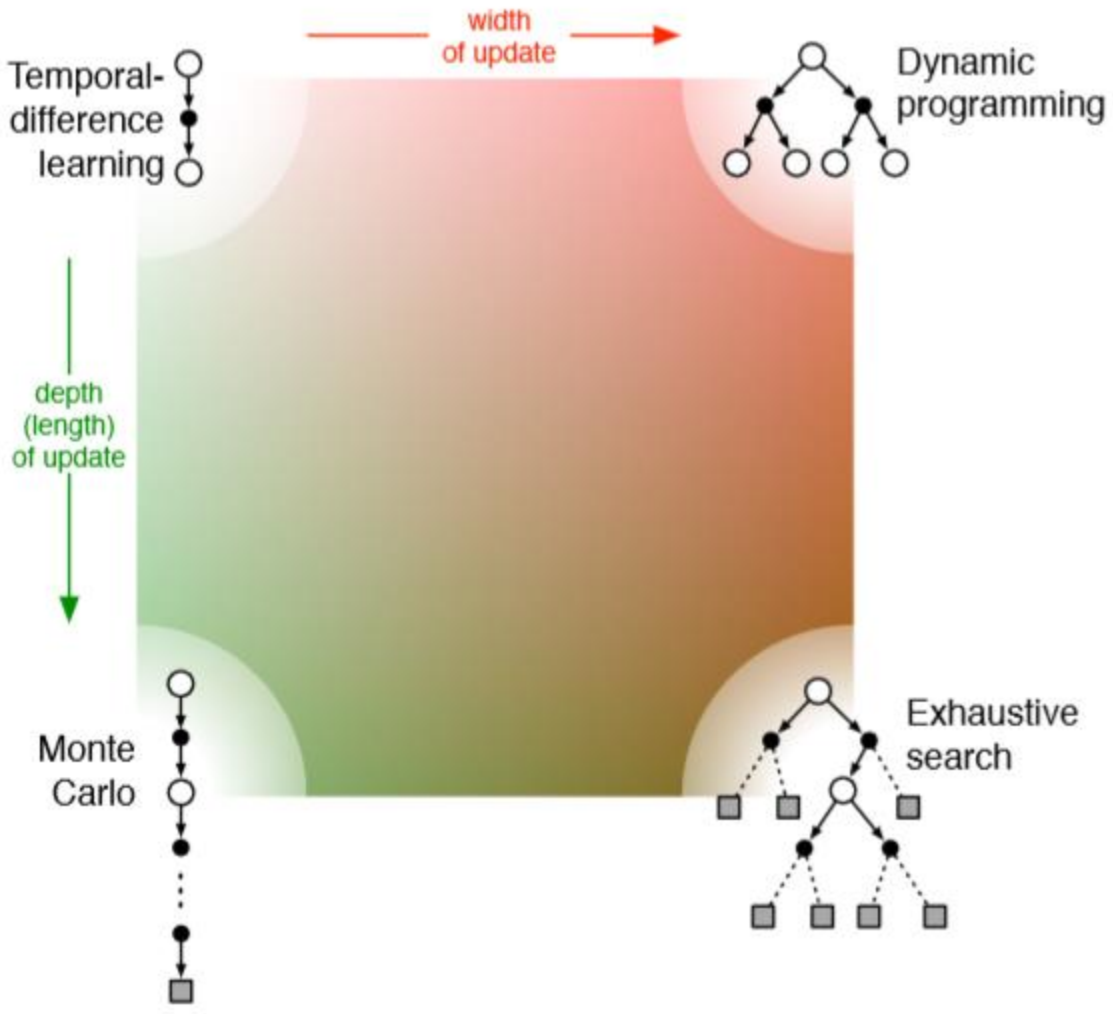
- Decision-time planning algorithms
- Uses MC control applied to simulated trajectories starting at current state
  - Average returns of many simulated trajectories with each possible action and then following *rollout* policy
  - When estimate accurate, highest value action is executed
- Unlike MC methods does not estimate complete optimal action-value function
  - Produces MC estimates of action values only for each current state and given policy
- Make immediate use of action-value estimates and then discard them
  - Generally no long-term memory of values and policies
- Maximizes estimate of Q for s and each action to improve upon rollout policy
  - Policy improvement theorem holds
  - Not looking for optimal policy

Monte-Carlo Tree Search

- Rollout algorithm BUT accumulating value estimates
- Estimates direct simulations toward highly-rewarding trajectories
- Executed at each new state to select agent's action just for that state
- Multiple focused simulations from current state to terminal state
  - Can be truncated as we have seen for RL
- Doesn't have to retain approximate V functions or policies
  - Although in many implementations it does (it's useful)
- Tree policy: select actions that look promising based on simulated trajectories
  - Could be eps-greedy or UCB selection rule
- No values are stored for states beyond the tree



Summary


width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search