SMU Lyle School of Engineering

Reinforcement Learning

Planning and Learning with Tabular Methods

Sutton/Barto* Chapter 8

Michael Hahsler, SMU

With figures from Sutton/Barto*

*Sutton and Barto, Reinforcement Learning: An Introduction, 2nd edition, MIT Press, Cambridge, MA, 2018

Topics of this Course

- Introduction to reinforcement learning
- Markov decision processes
- **Part I: Tabular Methods**
  - Dynamic programming
  - Monte Carlo methods
  - Temporal-difference learning
  - Multi-step bootstrapping
  - **Planning and learning with tabular methods**
- Part II: Approximate Solution Methods
  - Prediction and Control using Approximation
  - Eligibility Traces
  - Policy Gradient Methods
- Part III: Modern RL Methods
  - Deep Reinforcement Learning
  - Current Applications

Summary of Notation

General

X	capital letters: random variables
x, p	lower-case letters: realizations of random variables or scalar functions
$\boldsymbol{w}$	Bold lower-case letters: real-valued vectors (even if random variables)
$\boldsymbol{W}$	bold capitals: matrices
α	Greek letters: parameters (vectors if in bolt)
$\Pr\{X = x\}$	probability that a random variable X takes on the value x
$X \sim p$	random variable X selected from distribution $p(x) = \Pr\{X = x\}$
$\mathbb{E}[X]$	expectation of a random variable X, i.e., $\mathbb{E}[X] = \sum_x p(x)x$
$\text{argmax}_a f(a)$	a value of action a at which $f(a)$ takes its maximal value

Value Function

G_t	return (cumulative reward) following time t
$G_{t:h}$	return from t to h (discounted and corrected)
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or v_*

MDP

s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all (nonterminal) states, $\mathcal{S}^+$ are all states
$\mathcal{A}(s)$	set of all actions available in state s
γ	discount-rate parameter
t	discrete time step
T	final time step of an episode (a.k.a. horizon)
A_t	random variable for the action at time t
S_t	random variable for the state at time t
R_t	random variable for the reward at time t
$p(s', r \mid s, a)$	probability of transition to state s' and receiving reward r, from state s taking action a.
$p(s' \mid s, a)$	probability of transition to state s' fom state s taking action a.
$r(s, a)$	expected immediate reward from state s after action a.
$r(s, a, s')$	expected immediate reward from state s to s' with action a.
$\pi(a \mid s)$	probability of taking action a in state s under stochastic policy π
$\pi(s)$	action taken in state s under deterministic policy π

Planning vs. Learning

Planning vs. Learning

Model-based RL

- Requires knowledge of the MDP model.
- Methods: Dynamic Programming, heuristic search
- Use the known model to **plan**

Model-free RL

- Model is unknown.
- Methods: MC, TD
- **Learn** from interactions with the environment (sampling).

All methods:

- Compute a value function/policy.
- Look ahead to future events and compute a backed-up value to update an approximate value function.

What is a Model

Model of the environment: Anything the agent can use to predict the outcome of actions (e.g., next state and reward).

Models can be:

- **Distribution models**: Specify the complete distribution of all possible outcomes.
  - For finite MDPs this is the function $p(s', r \mid s, a)$
  - Used by dynamic programming methods for planning.

- **Sample models**: Simulate the real environment by producing one outcome for an action. The outcome needs to be a valid sample from the distribution model.
  - Are typically easier to create.
  - Can be used for MC and TD methods for learning. This is called trajectory sampling.

Notes:

- The real environment can be seen as a physical model with sequential sampling access.
- Model-free methods often use sample models when accessing the real environment is slow or expensive.

Can we integrate learning and planning?

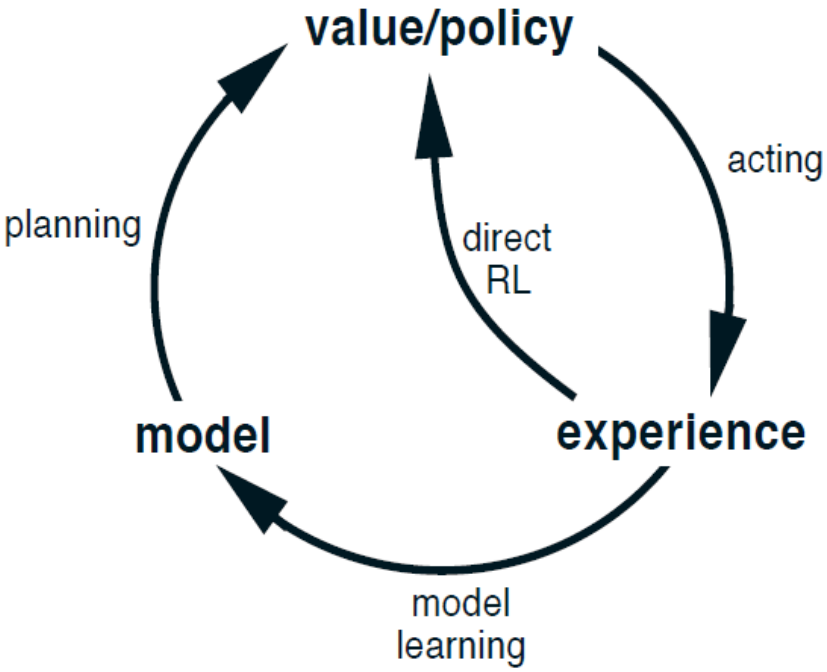
Dyna

Integrating Planning, Acting, and Learning

Integrating Planning, Acting, and Learning

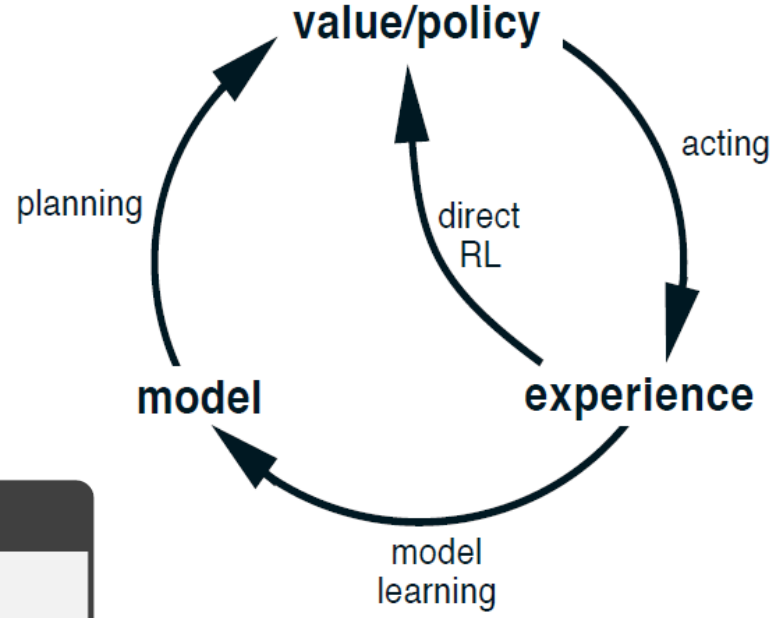
- **Direct RL is model-free Learning**:
  - Use the experience directly to improve the value function/policy. E.g., via one-step Q-learning.

- **Model Learning and Planning**:
  - Real experience is also used to learn a model. E.g., by recording for each $S_t, A_t \rightarrow R_{t+1}, S_{t+1}$ to estimate $\hat{p}(S_{t+1}, R_{t+1} | S_t, A_t)$
  - The model can be used for planning to improve the value function/policy. E.g., by sampling from the model to update the Q-values.



Advantage: Makes the approach much more sample efficient. Experience is not just used once but retained in the model.

Alg. Tabular Dyna-Q



Tabular 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) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Take action A; observe resultant reward, R, and state, S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment) — Model learning

(f) Loop repeat n times: — Planning

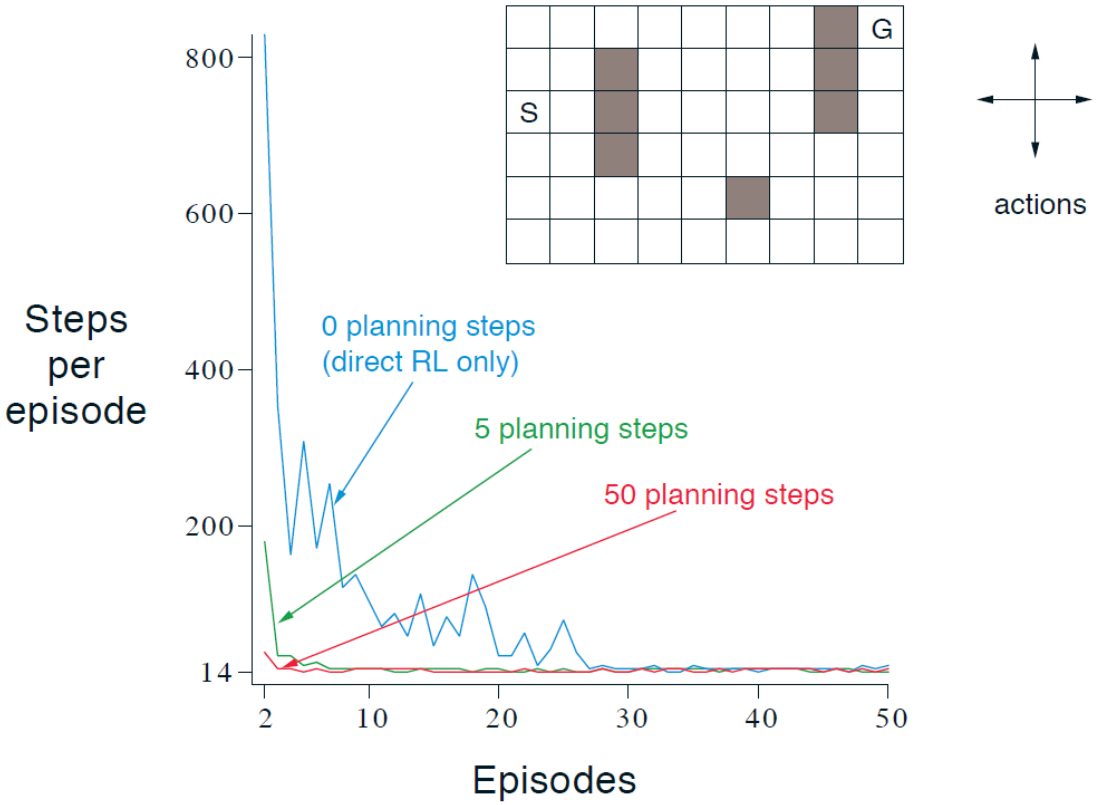
$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

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

The Effect of Planning



- **Benefit**: Adding planning speeds learning by reusing sampled experience.
- **Issues**: Planning is a problem if the model is wrong:
  - Based on an insufficient number of samples.
  - Function approximation not great.
  - Environment changes
- Dyna quickly corrects too optimistic predictions since they will be selected for the next real action.
- For changing environments:
  - Keep exploring.
  - Keep track of long action/state combinations that were not tried.

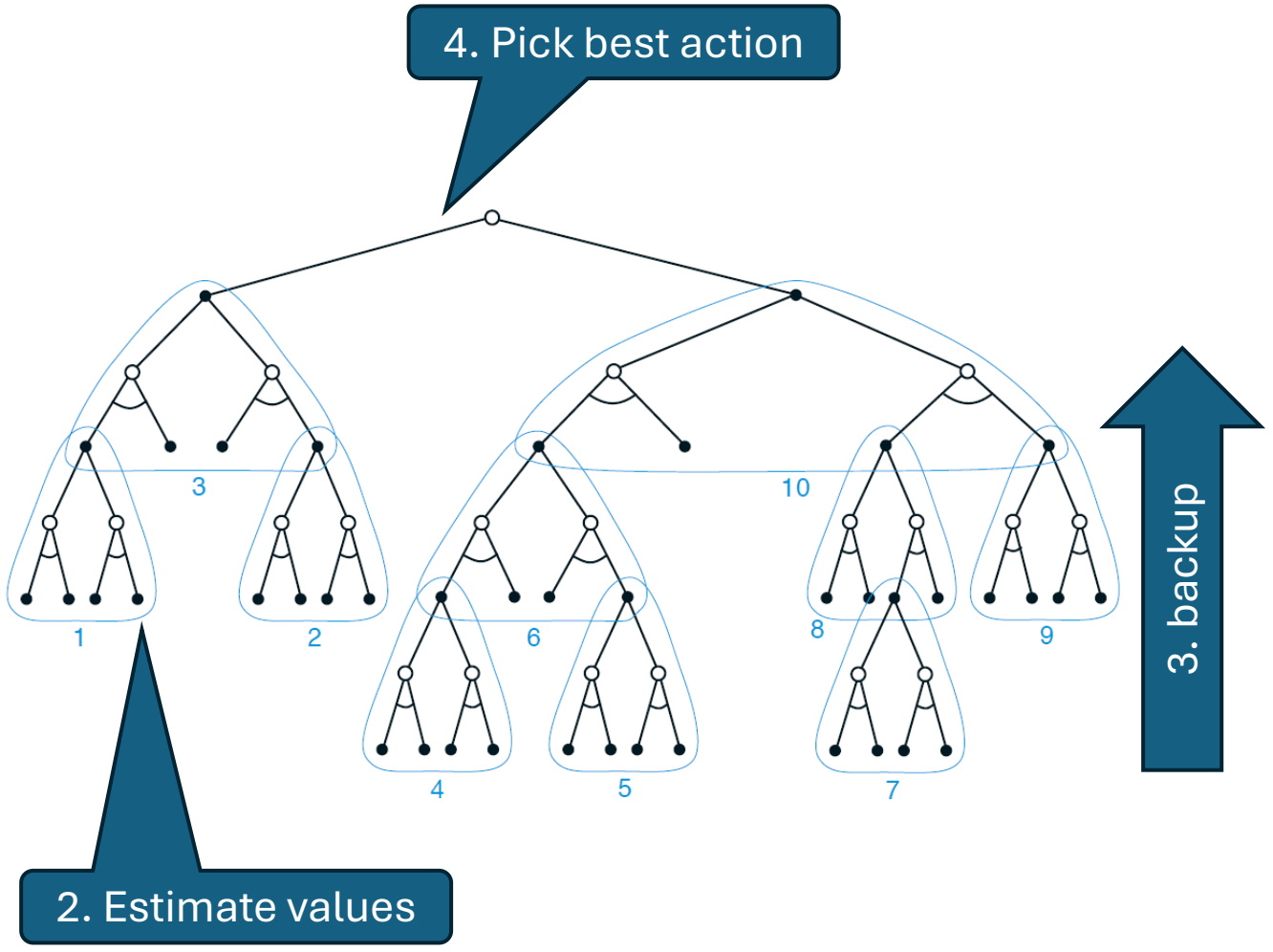
Heuristic Search and Rollout Algorithms

Decision-time planning

Option 1: Heuristic Search

At decision time in state s:

1. Consider a large tree of possible continuations (trajectories of a given length) for the current state.
2. Apply an approx. value function to the states in leaf nodes.
3. Back values towards the current state (using max).
4. Choose the best next action for the current state.



Note: This is like heuristic min-max search for games.

Option 2: Rollout Algorithms

Start with an initial policy π called the base policy.

At decision time in state s:

1. Monte Carlo Policy Estimation: Use many simulated trajectories to approximate all state/action values $q_\pi(s, a)$ for the current state s.
2. Choose the best action $a = \underset{a}{\operatorname{argmax}}\, q_\pi(s, a)$ according to the estimates. This is called the rollout policy.

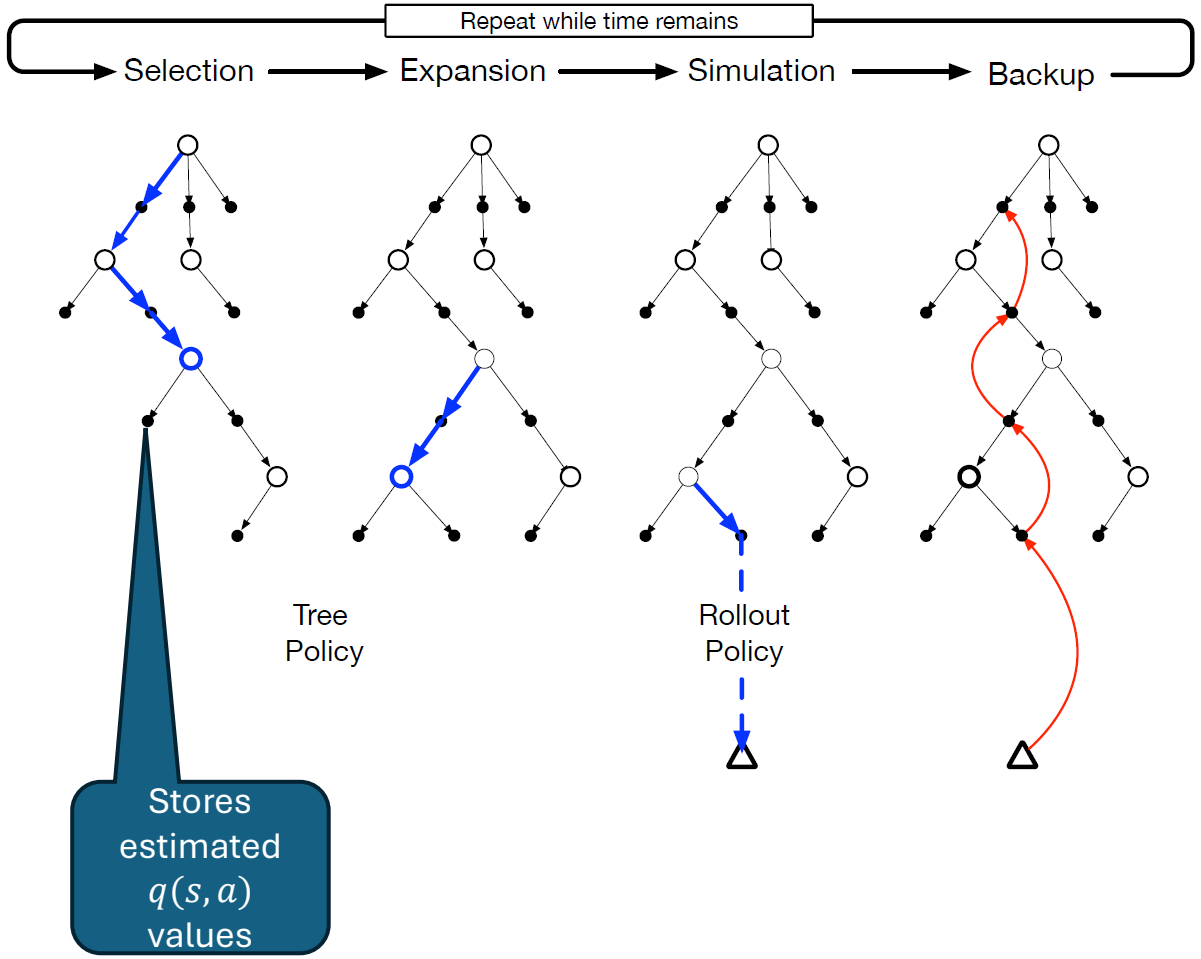
Properties:

- If the base policy is reasonable (e.g., a good heuristic), rollouts are locally optimal.
- Rollout policy is guaranteed to be at least as good as the base policy (in expectation)
- You get policy improvement without solving Bellman equations.

Rollout Algorithm: Monte Carlo Tree Search

- MCTS is a rollout algorithm that stores some intermediate estimates in a depth-limited tree structure.
  - **Selection**: Pick the currently best leaf node (e.g., greedy, UCB).
  - **Expansion**: Add new nodes, but keep the tree small enough.
  - **Simulation**: From a selected node, simulate complete episode(s).
  - **Backup**: update the tree with the result of the playout.

- **Advantage**: selection steers the simulation towards better trajectories. This avoids approximating the full value function.



MTCS can be seen as an RL method that selectively approximates q-values useful for choosing good actions.

What you Should Know

- Planning requires a model of the environment.
  - Distribution model
  - Sample model

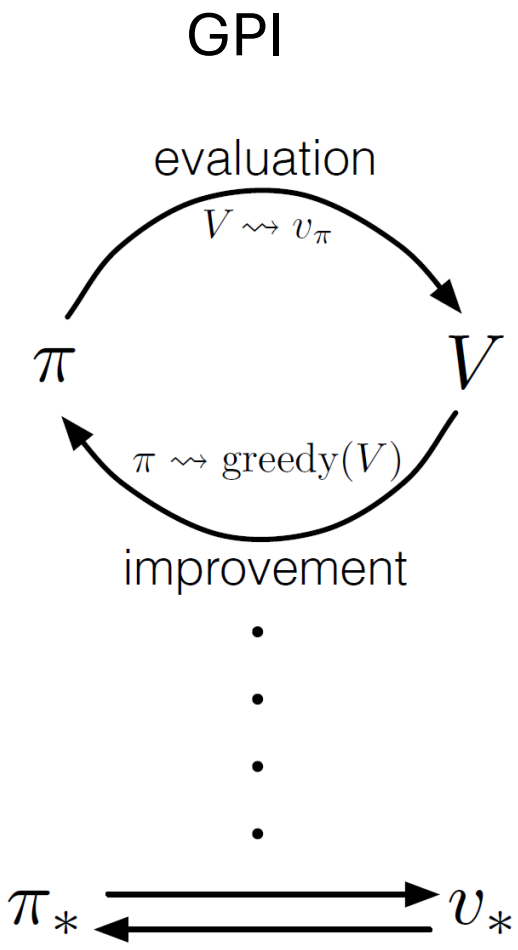
- DP requires a distribution model because it uses expected updates.
- Model-free RL, like TD, can use a sample model to perform sample updates.
- Decision-time planning can also use a sample model. E.g., rollout algorithms sample trajectories.

- Planning and learning are related: They try to estimate the same value function.

Summary of Part I: Tabular Solution Methods

Common Ideas

1. Estimate a value function.
2. Back up along actual or possible state trajectories.
3. Generalized policy iteration (GPI): maintain an approx. value function and an approx. policy. Each is used to update the other. Converges due to the policy improvement theorem.

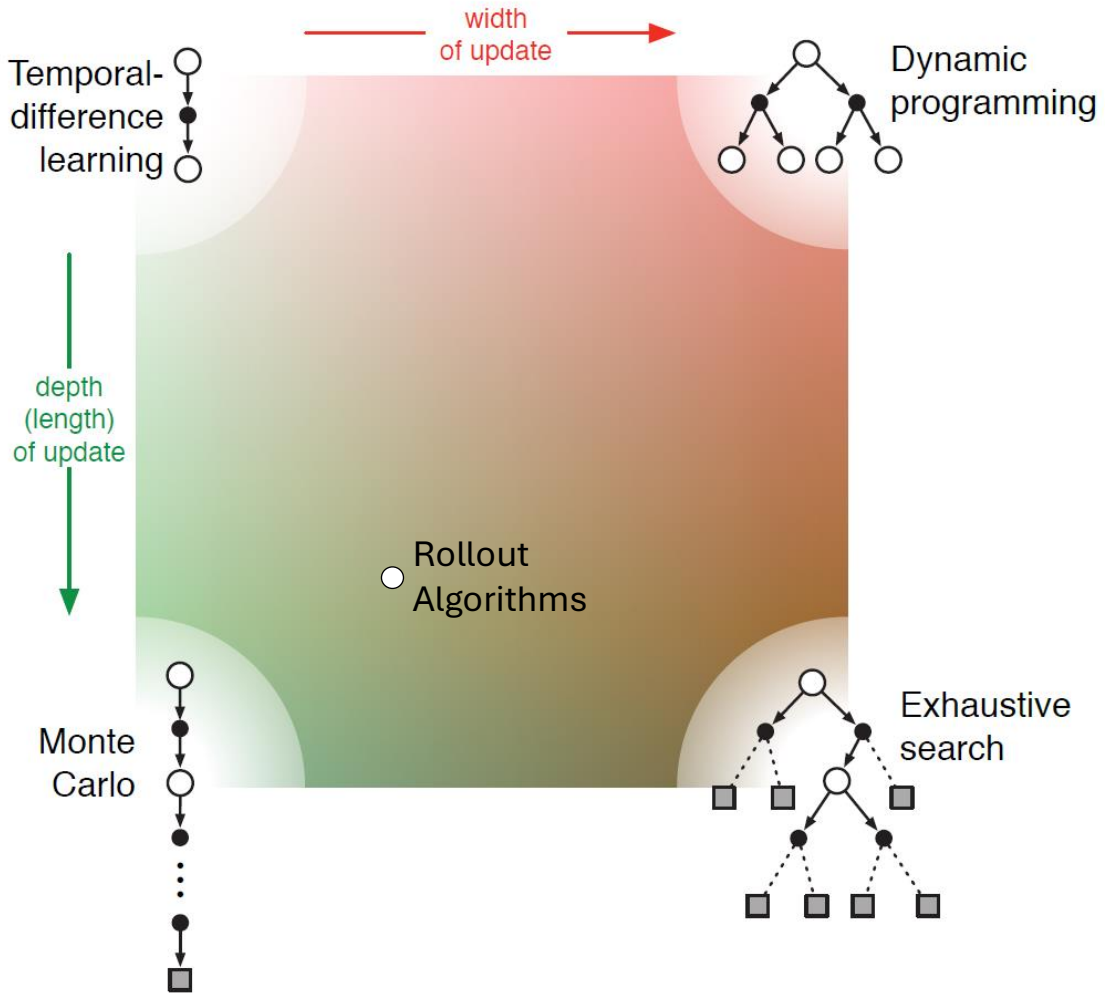


Update Depth vs. Update Width

One-step methods

n-step methods

Full-return methods



Sampled trajectory → Consider all actions

Each method is located somewhere in this continuum.

On-Policy vs. Off-Policy

On-Policy: Learn the policy the agent is following.

Off-Policy: Learn a target policy that is different from the followed behavior policy.

Off-Policy is useful because:

- Learn optimal policies while exploring
- Reuse of data: sample efficiency, offline, and batch RL
- Safe learning and exploration

Off-Policy learning is harder because:

- Requires distributional correction: E.g., importance sampling
- May suffer from high variance (due to importance sampling)
- Instability with non-linear function approximation (deep RL)

Other Dimensions

- **Definition of return**: Episodic vs. continuing task; discounted vs. undiscounted.
- **Action values vs. state values**
- **Exploration vs Exploitation**: This is a trade-off. Many methods need exploration for convergence.
- **Synchronous vs. Asynchronous**: Are all states updated at once or one by one in some order?
- **Real vs. Simulated**: Are real experiences, simulations, or both used for updates?

Issues with Tabular Methods

1. Tables need much **space**. A complete table with q-values has size
$$O\left(|\mathcal{S}| \times |\mathcal{A}|\right)$$
The state space is typically extremely large, or even infinite, in continuous spaces.

2. We need a lot of **data** to estimate all entries.

3. Two states may be very similar and therefore should have similar q-values. Tabular methods **cannot generalize across states**.

Next, we will talk about **approximate solution methods** to deal with these issues.