SMU Lyle School of Engineering

# Reinforcement Learning

## Dynamic Programming

Sutton/Barto* Chapter 4

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 |
| $\alpha$ | 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 $\pi$ (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 $\pi$ |
| $q_*(s, a)$ | value of taking action $a$ in state $s$ under the optimal policy |
| $V, V_t$ | array estimates of state-value function $v_\pi$ or $v_*$ |
| $Q, Q_t$ | array estimates of action-value function $q_\pi$ 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 $s$ |
| $\gamma$ | 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 $\pi$ |
| $\pi(s)$ | action taken in state $s$ under deterministic policy $\pi$ |

# Goal

- **Remember**: Finite MDPs have an optimal deterministic policy (under some mild conditions).
- **Goal**: Compute the **optimal deterministic policy** $\pi_* = \underset{a}{\operatorname{argmax}}\, q_*(s,a)$ given a perfect MDP model.
- Solve the Bellman optimality equations (the optimal policy always picks the best action):

$$v_*(s) = \max_a \mathbb{E}_\pi \left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right]$$

or

$$q_*(s,a) = \max_a \mathbb{E}_\pi \left[R_{t+1} + \gamma \underbrace{\max_{a'} q_*(S_{t+1}, a')}_{= v_*(S_{t+1})} \middle| S_t = s, A_t = a\right]$$

- **Issues**: The value estimate for $S_t$ depends on the current estimates of the successor states $S_{t+1}$! We need to solve all optimality equations at the same time.

**Control Problem:**
Find the optimal policy

- **Idea**: We can find the optimal policy by stepwise **improving a policy** till it is optimal. A policy is better if it has larger state values.

$$\pi' \geq \pi \Leftrightarrow v_{\pi'}(s) \geq v_\pi(s) \qquad \text{for all } s \in \mathcal{S}$$

- To do this we need to be able to compute $v_\pi$ and $v_{\pi'}$.

**Prediction Problem:**
Estimate the value function for a policy.

# Tabular Methods

- Tabular means: The value functions $v$ or $q$ are stored in arrays (tables) called $V$ or $Q$.
- Important property: Entries in the table do not influence each other. We can change them independently.
- Tabular methods find the exact solution (i.e., optimal value function and policy) given enough memory and time.

- **Issues**:
  - Table size:
    - $v$ has $|\mathcal{S}|$ entries
    - $q$ has $|\mathcal{S}| \times |\mathcal{A}|$ entries

    This is only feasible for problems with a small discrete state/action space!
  - No generalization: Knowledge about the value of one state (state-action pair) does not help us with similar states

- We will later talk about approximate methods that generalize and can work with large state spaces.

Value Function

| $s$ | State Value $V(s)$ |
|---|---|
| (1,1) | 0.7453 |
| (1,2) | 0.8016 |
| … | … |

Q-Table

| $s$ | $a$ | $q(s,a)$ |
|---|---|---|
| (1,1) | up | 0.7453 |
| (1,1) | right | 0.6709 |
| (1,1) | down | 0.7003 |
| (1,1) | left | 0.7109 |
| … | … | … |

or

| | up | right | down | left |
|---|---|---|---|---|
| (1,1) | 0.7453 | 0.6709 | 0.7003 | 0.7109 |
| … | … | … | … | … |
| … | … | … | … | … |

# Prediction vs. Control

- In this class, we will always talk about two tasks:

1. **Prediction to evaluate a policy**: Estimate the value function for a given policy. We can use this to compare two policies.

2. **Control**: Control the agent's behavior to find the optimal policy. This requires being able to evaluate policies.

# Prediction: Policy Evaluation

What is the value function for a given policy?

# Policy Evaluation to Estimate $v_\pi$

**Goal**: Compute the state-value function $v_\pi$ for a given policy $\pi$.

Bellman expectation equations give us the recursive relationship:

$$\begin{aligned} v_\pi(s) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1}|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r\,|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

with $|\mathcal{S}|$ equations, one for each state $s$.

**Guarantees**:

- Discounted finite MDPs ($\gamma < 1$ ) have a unique $v_\pi$.
- Undiscounted finite MDPs ($\gamma = 1$) have a unique $v_\pi$ if $\pi$ guarantees to reach a terminal state.
- Finite-horizon MDPs have unique $v_\pi^k$ where $k$ is the time step. I.e. , value depends on how close we are to the end of the episode.

# Solution Option 1: Linear Program

Solve the system of $|\mathcal{S}|$ linear Bellman Expectation Equations:

$$v_\pi(s_1) = \sum_a \pi(a|s_1) \sum_{s',r} p(s', r\,|s_1, a)[r + \gamma v_\pi(s')]$$

$$v_\pi(s_2) = \sum_a \pi(a|s_2) \sum_{s',r} p(s', r\,|s_2, a)[r + \gamma v_\pi(s')]$$

$$\ldots \text{ for all } s \in \mathcal{S}$$

Formulate as a linear program

- Objective function: $\min_V \sum_{s\in\mathcal{S}} V(s)$
- Constraints: $V(s) \geq \sum_{s',r} p\big(s', r|s, \pi(s)\big)[r + \gamma V(s')] \qquad \forall s \in \mathcal{S}$

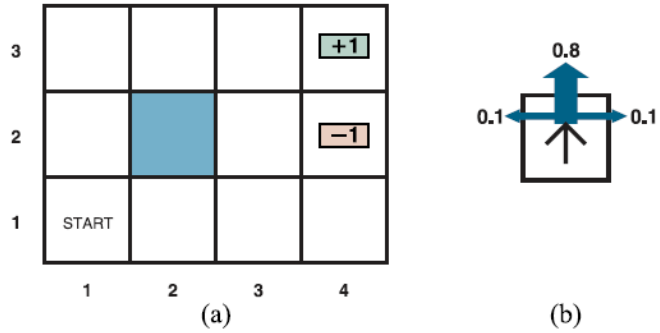
Why this works: The smallest feasible solution will satisfy all Bellman equations.

$$V(s) = \sum_{s',r} p\big(s', r|s, \pi(s)\big)[r + \gamma V(s')] \qquad \forall s \in \mathcal{S}$$

# Solution Option 2: Iterative Policy Evaluation

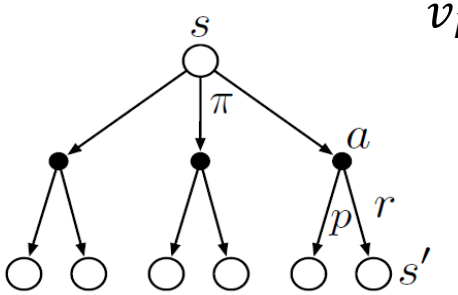
**Dynamic Programming**: solves a complex optimization problem by breaking it down into smaller, overlapping subproblems and using their solutions to build the overall optimal solution.

Sweep the state space




- Start with arbitrary values for all $v(s)$
- Update all state values (called sweeping over the state space) using the **Bellman Update** ($k$ is the iteration counter):

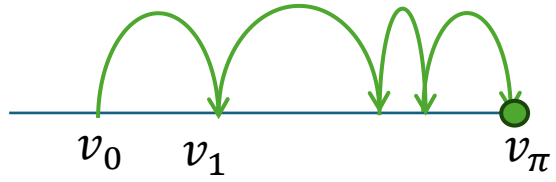
Backup diagram




$$v_{k+1}(s) \stackrel{\text{def}}{=} [R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r\,|s, a)[r + \gamma v_k(s')]$$

In DP called "expected update."
Uses the expectation over all possible actions and next states.

Bellman updates




Distance means difference

- **Convergence**: Updates converge to $v_\pi$ for $k \to \infty$ iterations. Reason is on the next slide.

# Convergence of Iterative Policy Evaluation

- The Bellman update can be rewritten as an operator:
  **Bellman Expectation Operator** $T^\pi$:

$$(T^\pi v)(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r\,|s, a)[r + \gamma v(s')]$$

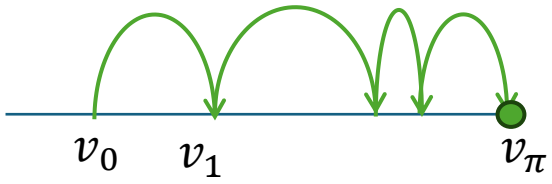
Notation:
$(T^\pi v)$ is the new value function after applying the operator. We could write $v_{k+1} = T^\pi(v_k)$

Note:
$$v_\pi = T^\pi(v_\pi)$$

- The operator $T^\pi$ is $\gamma$-contracting with $v_\pi$ as its fixed point:
  (see textbook for derivation)

$$\|v_\pi - T^\pi v\|_\infty \leq \gamma \|v_\pi - v\|_\infty \quad \text{for any} \quad \gamma < 1$$

Bellman updates




Distance means difference

This means the largest difference (infinity norm) will be reduced by applying $T^\pi$, leading to convergence to $v = v_\pi$ when is applied many times.

Convergence is guaranteed if the discount factor is strictly smaller than 1!

# Alg. Iterative Policy Evaluation

Notation: $V$ is the tabular estimate of $v$

**Iterative Policy Evaluation, for estimating $V \approx v_\pi$**

Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Bellman operator $T^\pi$:
$V_{k+1} = T^\pi V_k$

Stop when value function changes little:
$\|T^\pi V_k - V_k\|_\infty < \theta$
It can be shown that the error is bounded:
$\|V_k - v_\pi\|_\infty < \frac{\theta}{1-\gamma}$

If we stop policy evaluation early (after $n$ iterations) then it is called **modified policy iteration** and we end up with an approximation.

# Control: Policy Iteration

Find the optimal policy based on the environment model.

# Policy Improvement and the Greedy Policy

- **Goal**: Find a better policy.

**Policy Improvement Theorem**

$$q_\pi\big(s, \pi'(s)\big) \geq v_\pi(s) \;\; \forall s \in \mathcal{S} \quad \Rightarrow \quad v_{\pi'}(s) \geq v_\pi(s) \;\; \forall s \in \mathcal{S}$$

If $\pi'$ is the same as $\pi$ but has a better action for at least one state then $\pi'$ is strictly better than $\pi$

- **Policy improvement**: creating a new policy $\pi'$ by choosing in each state the best (aka greedy) action with respect to $q_\pi(s, a)$.

$$\pi'(s) = \operatorname*{argmax}_a q_\pi(s, a)$$

- **Guarantee**: This **greedy policy** $\pi'$ is strictly better unless $\pi$ is already optimal.

# Policy Iteration

**Idea**: Create a sequence of monotonically improving policies:

$$\pi_0 \overset{E}{\rightarrow} v_{\pi_o} \overset{I}{\rightarrow} \pi_1 \overset{E}{\rightarrow} v_{\pi_1} \overset{I}{\rightarrow} \pi_2 \overset{E}{\rightarrow} v_{\pi_2} \overset{I}{\rightarrow} \pi_3 \overset{E}{\rightarrow} v_{\pi_3} \overset{I}{\rightarrow} \dots \pi_* \overset{E}{\rightarrow} v_*$$

$\overset{E}{\rightarrow}$ policy eval. step
$\overset{I}{\rightarrow}$ policy improvement step

**Guarantees**:

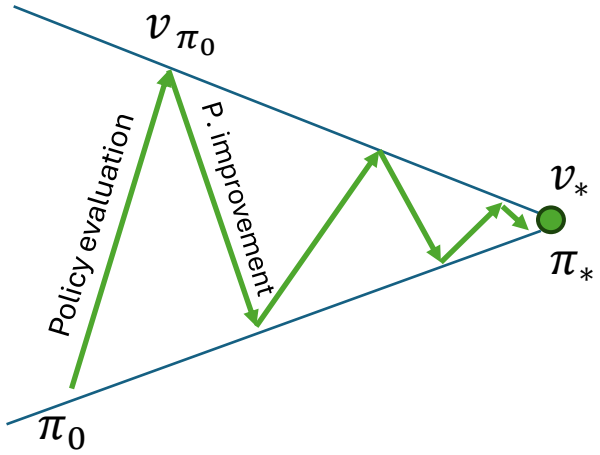
- Policy improvement is guaranteed to improve the policy.
- Finite MDPs have a finite number of policies → Policy Iteration must **converge**!

**Advantages**:

- Policy becomes stable before $v_*$ is reached.
- Policy iteration typically converges fast.

**Issue**:

- Policy estimation is computationally expensive!
  Using fewer iterations leads to modified policy iteration.



# Alg. Policy Iteration

**Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$**

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      *old-action* $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Just policy evaluation from before. We can also stop this loop early (modified policy iteration).

Picks the greedy action with the highest Q-value to improve the policy.

# Control: Value Iteration

Find the optimal value function based on the environment model.

# Value Iteration

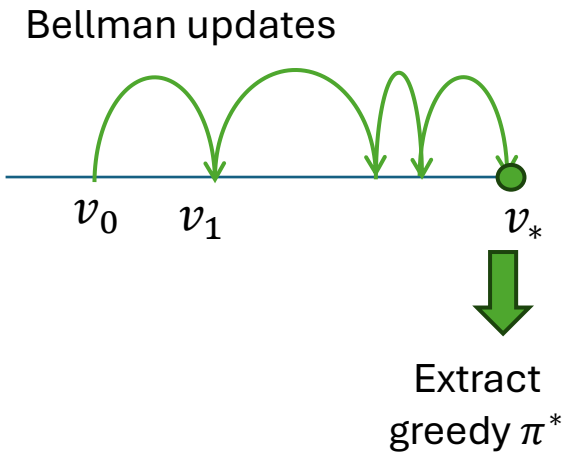
- Combines policy improvement and truncated policy evaluation into a single step. This is called the **expected Bellman optimality update**.

$$v_{k+1}(s) \stackrel{\text{def}}{=} \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r\,|s, a)[r + \gamma v_k(s')]$$

$\max_a$ ... policy improvement
$\mathbb{E}$ ... truncated policy evaluation (1 step)

- Written as the **Bellman optimality operator**:

$$(T^* v)(s) = \max_a \sum_{s',r} p(s', r\,|s, a)[r + \gamma v(s')]$$




- **Guarantee**: $T^*$ is a non-linear operator, but it is also $\gamma$-contracting. The Sequence $\{v_k\}$ converges to $v_*$.

# Alg. Value Iteration

**Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$**

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(terminal) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(\right.$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta <$ [illegible]

3. Policy In[illegible]
   *policy-sta*
   For each
   *old-action* $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma\right.$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$
   If *policy-stable*, then stop and return $V \approx v_*$

Combine improvement and modified evaluation (1 iteration only) into a single update.

**Value Iteration, for estimating $\pi \approx \pi_*$**

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Bellman optimality operator: $V_{k+1} = T^* V_k$

Stopping criterion from policy evaluation

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

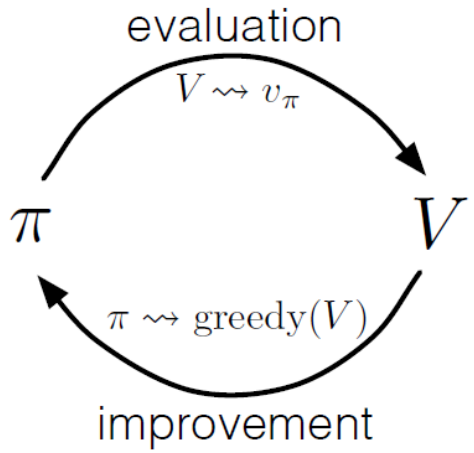
Extract the greedy policy.

Code...

# GPI: Generalized Policy Iteration

Generalizing value and policy iteration into a single framework.

# GPI: Generalized Policy Iteration

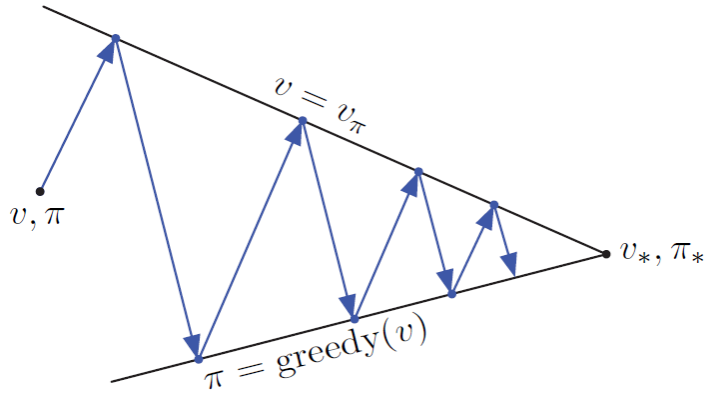



**Steps**

- **Policy evaluation**: make the value function consistent with the current policy $\pi$.
- **Policy improvement**: make policy greedy with respect to the current value function estimate $V$.

Most DP methods can be described as a special case of GPI:

- **Policy iteration**: each evaluation and improvement step is completed (by a sweep over $\mathcal{S}$) before moving on.
- **Value iteration**: uses a single iteration for the policy evaluation step.
- **Asynchronous DP**: Instead of sweeps over $\mathcal{S}$, updates states in some other order (e.g., focus on states where the value function changed a lot in the last step). This interleaves pol. eval. and pol. Improvement.




All three approaches typically converge to the optimal solution.

# Issues with DP Methods

**Require perfect knowledge of the model!**

## Space Complexity

- Tabular method: stores the complete value function as tables $V$ or $Q$.

## Time Complexity

- Updates repeatedly sweep over the whole state space.
- Wastes computation by updating states that are not relevant for the optimal behavior. I.e., the optimal policy will never visit them.
- Tabular methods can not generalize between similar states.

Asynchronous DP can help

# Efficiency

- **Direct Search**: In each state we can choose one of the available actions. Search space $O(|\mathcal{A}|^{|\mathcal{S}|})$ is **exponential** in the number of states $|\mathcal{S}|$.

- **LP (linear program):** Linearize the non-linear Bellman Optimality Equations by introducing constraints for all action/state pairs. Leads to a time complexity of $O(|\mathcal{S}|^3)$.

- **DP**: Time $O(|\mathcal{S}|^2 \times |\mathcal{A}|)$ is **quadratic** in $|\mathcal{S}|$.

Sparse transition matrices lead to better performance.

**Conclusion**: Search and LP are impractical. The number of actions is typically relatively small. **DP can be used today with millions of states.**

# What you need to know

**Requirements**:

- **Model access**: DP methods need access to the **model**'s function $p$.
- **Small discrete state/action space** since it is a tabular method.

**Tasks**:

- **Prediction**: Calculates the value function for a given policy called in DP policy evaluation.
- **Control**: Calculates the optimal policy for an MDP. Is not truly control, but an offline planning method.

**Approach**:

- Convert the recursive Bellman equations into the Belmann update operators and sweep the state space till the optimal solution is found.
  - Expected Bellman update operator $T^{\pi}$ vs. Bellman optimality operator $T^{*}$
  - Value iteration is a special case of modified policy iteration with a single policy evaluation iteration.
  - Convergence requires either $\gamma < 1$ or a guarantee that the policy will reach a terminal state.

**Generalized Policy Iteration (GPI)**:

- All DP methods can be generalized using the GPI evaluation/improvement framework.