Dynamic Programming

Benjamin Chew

What is Dynamic Programming?

Dynamic	sequential or temporal component to the problem
Programming	optimising a “program”, i.e. a policy

- Algorithms used to compute optimal policies given a perfect model of the environment as a Markov Decision Process (MDP)
- A method for solving complex problems
  - Break them down into sub-problems
  - Solve the sub-problems
  - Combine solutions to sub-problems

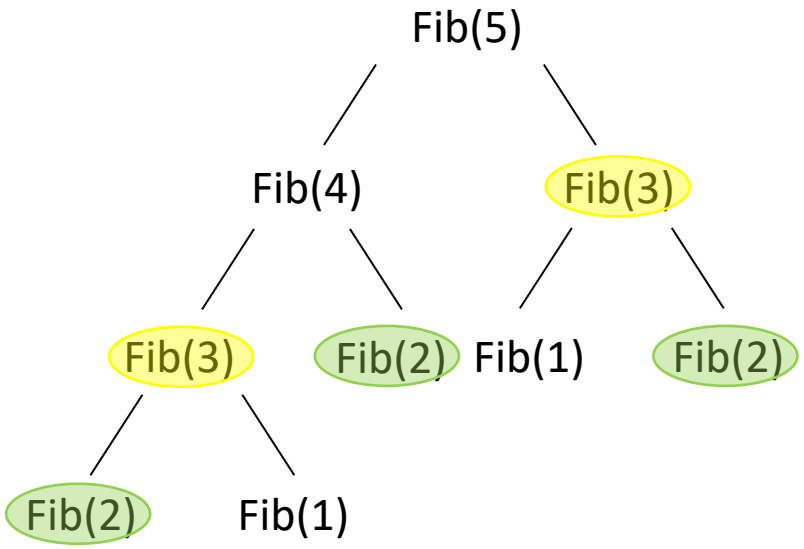
Requirements for Dynamic Programming

- Dynamic Programming is a very general solution method for problems which have two properties:
  - Optimal sub-structure
    - Principle of optimality applies
    - Optimal solution can be decomposed into sub-problems
  - Overlapping sub-problems
    - Sub-problems recur many times
    - Solutions can be cached and reused

- MDPs satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

Fibonacci Example

- 1, 1, 2, 3, 5, 8…



Policy Evaluation

Policy Evaluation: for a given policy π, compute the state-value function v_π

Recall: **State-value function for policy π**

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Recall: **Bellman equation for v_π**

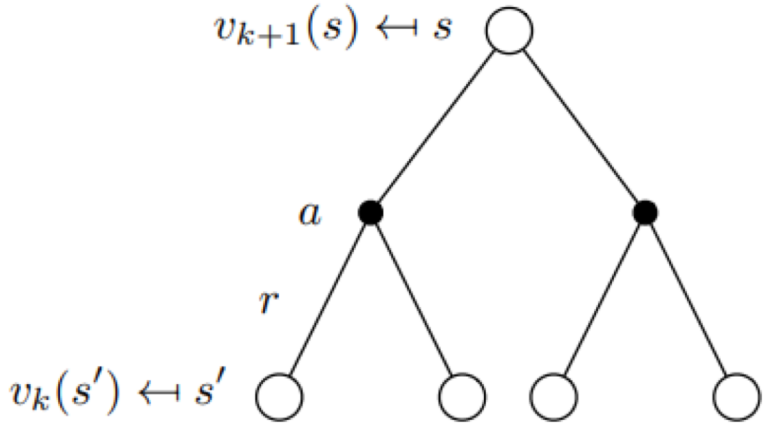
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]$$

Iterative Methods

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k \rightarrow v_{k+1} \rightarrow \cdots \rightarrow v_\pi$$

a "sweep"

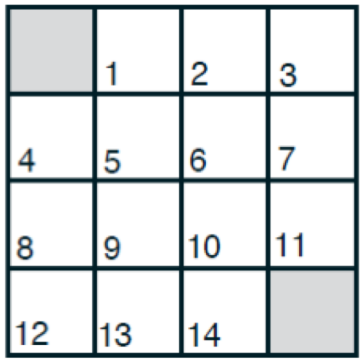
A sweep consists of applying a **backup operation** to each state.



$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big]$$

A Small Gridworld

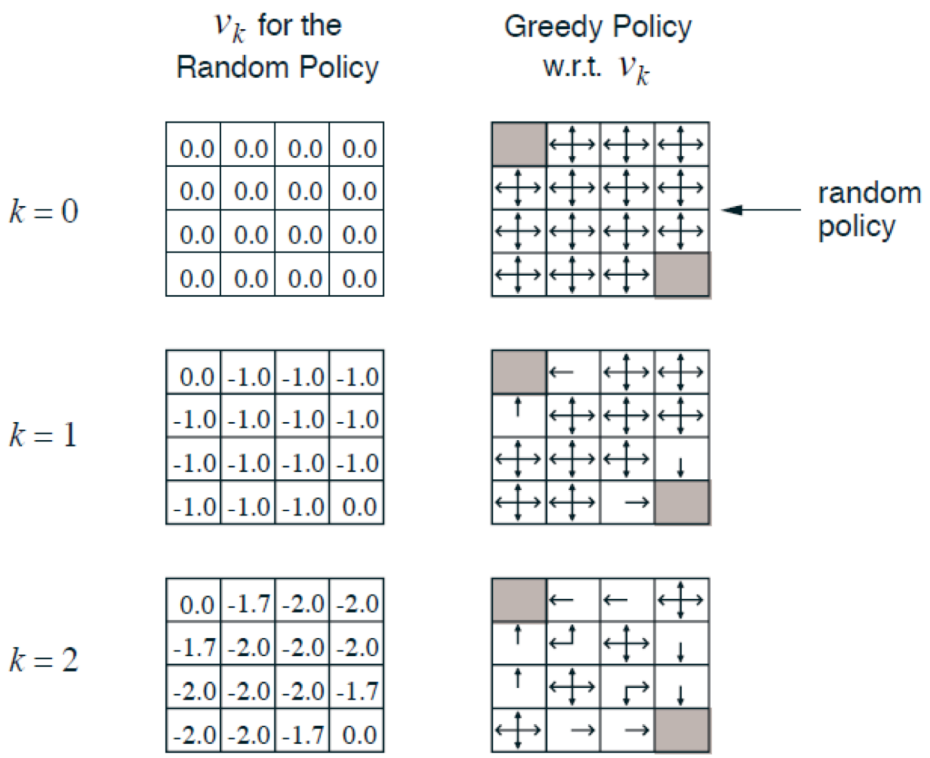
actions



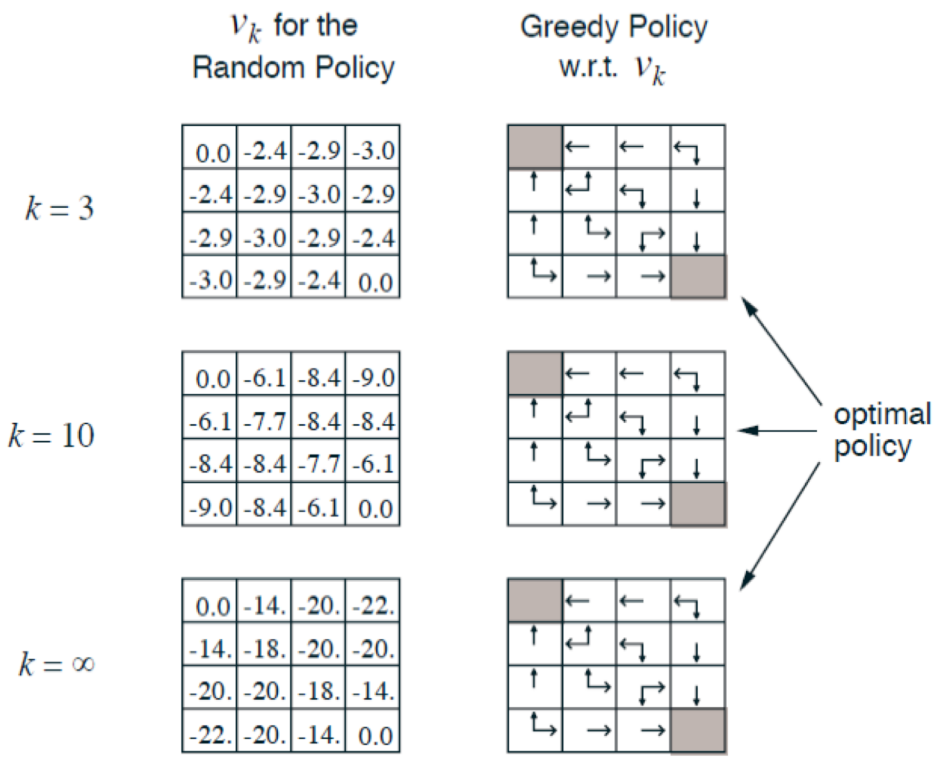
$R_t = -1$
on all transitions

- Undiscounted episodic MDP
- Nonterminal states: 1, 2, …, 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached

A Small Gridworld: Iterative Policy Evaluation



A Small Gridworld: Iterative Policy Evaluation



Policy Improvement

Suppose we have computed v_π for a deterministic policy π.

For a given state s,
would it be better to do an action $a \neq \pi(s)$?

It is better to switch to action a for state s if and only if

$$q_\pi(s,a) > v_\pi(s)$$

And, we can compute $q_\pi(s,a)$ from v_π by:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

Policy Improvement

Do this for all states to get a new policy $\pi' \geq \pi$ that is **greedy** with respect to v_π :

$$\begin{aligned}
\pi'(s) &= \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}$$

Policy Improvement

- This improves the value from any state *s* over one step,

$$q_\pi(s, \pi'(s)) = \max q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s\right] \\
&= v_{\pi'}(s).
\end{aligned}$$

Policy Improvement

- If improvements stop,

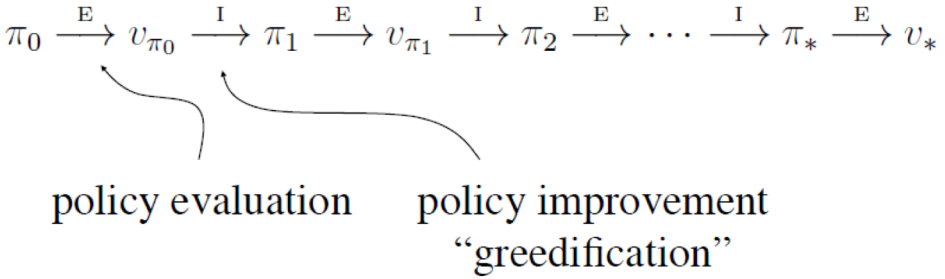
$$q_\pi(s, \pi'(s)) = \max q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

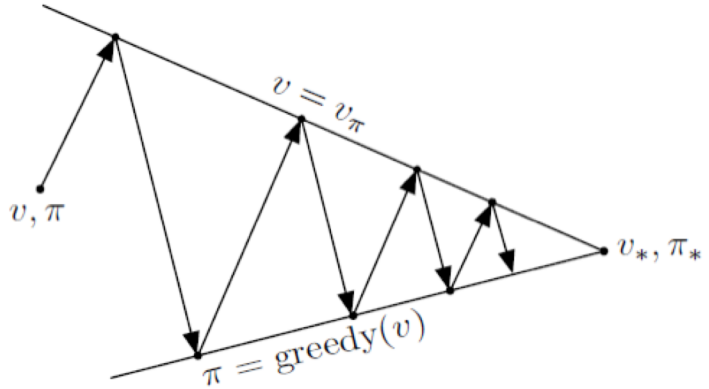
- Then the Bellman optimality equation has been satisfied

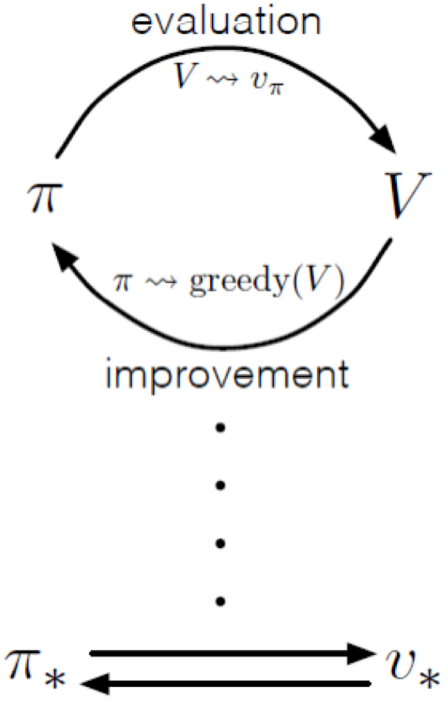
$$v_\pi(s) = \max q_\pi(s, a)$$

Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
so π is an optimal policy

(Generalized) Policy Iteration

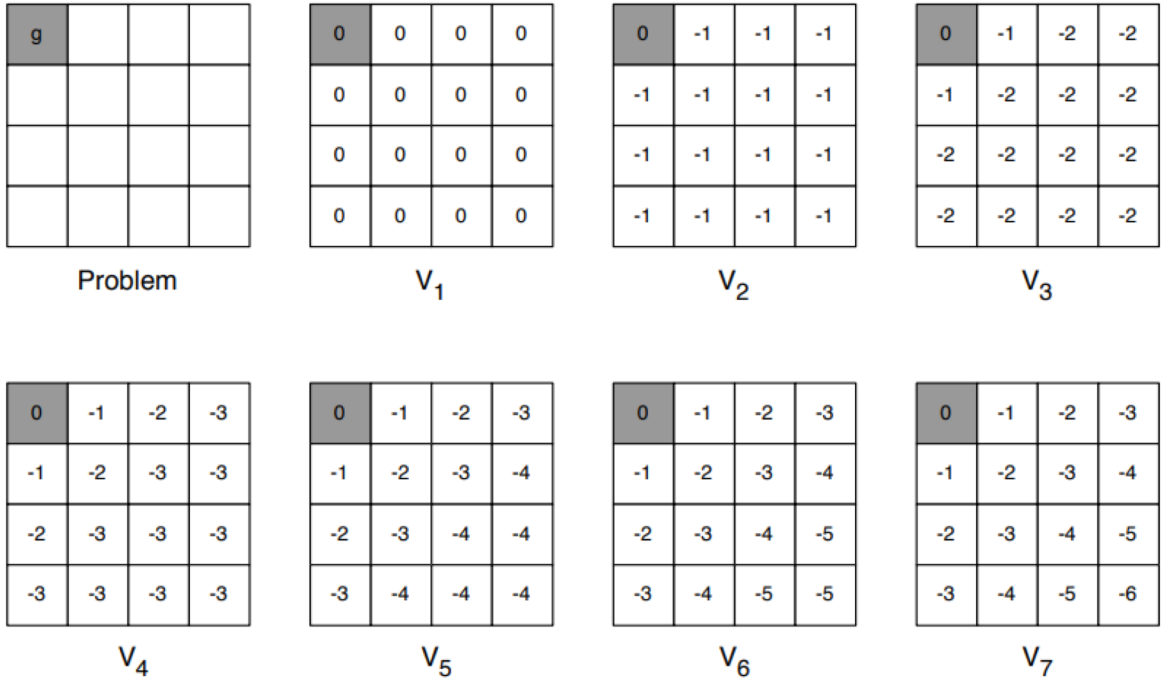






Value Iteration

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$



Value Iteration

- Problem: Find optimal policy π
- Solution: Iterative application of Bellman optimality equation

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

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

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 $\Delta < \theta$

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

Asynchronous Dynamic Programming

- Synchronous backups used for the DP methods described so far
  - Require exhaustive sweeps of the state set
- Asynchronous DP updates state values in any order
  - Repeat until convergence criterion is met:
    - Pick a state at random (or by design) and apply the appropriate backup
- Does not necessarily mean less computation

Efficiency of Dynamic Programming

- Time taken by DP methods to find an optimal policy is polynomial in the number of states and actions
- Curse of Dimensionality: Number of states grow exponentially with state variables
- In practice, DP methods can be applied to MDPs with millions of states
- Asynchronous DP can be applied to larger problems