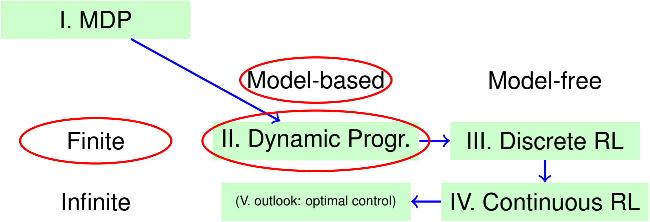
AIC/RL – Dynamic Programming (Part II)

Freek Stulp

Université Paris-Saclay

Where are we?


I. MDP
Model-based
Model-free
Finite
II. Dynamic Progr.
III. Discrete RL
Infinite
(V. outlook: optimal control)
IV. Continuous RL

Model-based vs. Model-free

- Environment as an MDP: $\{S, A, \mathcal{P}, \mathcal{R}\}$ (and $\{T, \mathcal{I}\}$)
  - S Possible states
  - A Possible actions
  - $\mathcal{P}$ Transition function
  - $\mathcal{R}$ Reward function
- Model-free
  - Agent knows S, A, but not $\mathcal{P}, \mathcal{R}$
- Model-based
  - Agent also knows $\mathcal{P}$, and perhaps also $\mathcal{R}$

- If agent doesn't completely know S: POMDPs
- If agent doesn't completely know A: ???

Outline

1. Values
2. Bellman Equation
3. Dynamic Programming
4. TP

Values (SuBa3.7)

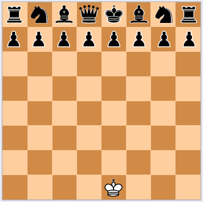
- What is the value of a state?
  - Informally: "*How good is it to be in a certain state?*"
  - Formally: $V^{\pi}(s)$ is expected return when starting in s and following π

Example: values in chess (Win=1.0, Loose=0.0, Draw=0.5)

You V=0.5

Me V=0.5

You V=1.0



Me V=0.0

Deep Blue V=1.0

Me V=0.0

- Value depends on the **state**
- Value depends on the **policy**

Values (SuBa3.7)

- What is the value of a state?
  - Informally: "*How good is it to be in a certain state?*"
  - Formally: $V^\pi(s)$ is expected return when starting in s and following π

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \qquad \text{Return (SuBa3.4)} \tag{1}$$

$$V^\pi(s) = \mathrm{E}_\pi \{R_t | s_t = s\} \qquad \text{Value (SuBa3.7)} \tag{2}$$

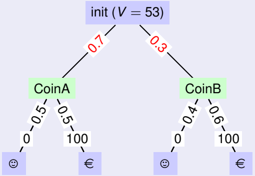
$$= \mathrm{E}_\pi \left\{ \sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s \right\} \tag{3}$$

- R_t is an <u>actual observation</u>, $V^\pi(s)$ is an <u>expectation</u>.
- $V^\pi(s)$ depends on future actions, i.e. which the policy will decide.

Values

Flipping coins example

- Choose one of two coins randomly
  - CoinA is fair, i.e. 50%/50%
  - CoinB is loaded, tails 60% of the time
- If you get tails you get 100, if heads 0
- Policy: choose CoinA 70% of the time



- What is the value, i.e. the expected return, of this game?

$$
\begin{aligned}
V^{\pi}(s) &= \mathrm{E}_{\pi}\left\{R_t | s_t = s\right\} && (4)\\
V^{\pi}(init) &= 0.7(0.5 \cdot 0 + 0.5 \cdot 100) + 0.3(0.4 \cdot 0 + 0.6 \cdot 100) && (5)\\
&= 0.7 \cdot 50 + 0.3 \cdot 60 && (6)\\
&= 53 && (7)\\
V^{\pi}(init) &= \sum_{a} \pi(init, a) \sum_{s'} \mathcal{P}^{a}_{ss'} \mathcal{R}^{a}_{ss'} && (8)\\
&= \pi(init, CoinA) \cdot (\mathcal{P}^{CoinA}_{init,\text{☺}} \cdot \mathcal{R}_{init,\text{☺}} + \mathcal{P}^{CoinA}_{init,\text{€}} \cdot \mathcal{R}_{init,\text{€}}) && (9)\\
&\quad + \pi(init, CoinB) \cdot (\mathcal{P}^{CoinB}_{init,\text{☺}} \cdot \mathcal{R}_{init,\text{☺}} + \mathcal{P}^{CoinB}_{init,\text{€}} \cdot \mathcal{R}_{init,\text{€}}) && (10)
\end{aligned}
$$

Values: Gridworld example

- Reward of 100 for going to `T`, -1 otherwise
- Simple case: deterministic policy and MDP, discount=1

$\pi(s)$

T	<	<	<
∧	<	<	<

$V^\pi(s)$

T	100	99	98
100	99	98	97

$\pi(s)$

T	<	<	<
>	>	>	∧

$V^\pi(s)$

T	100	99	98
94	95	96	97

$\pi(s)$

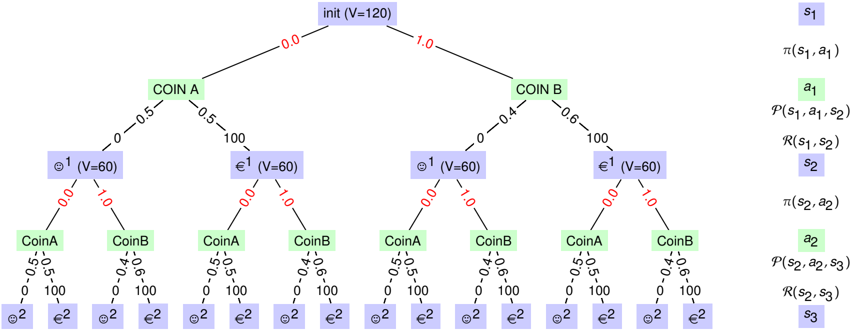
T	<	>	∨
∧	<	<	∧

$V^\pi(s)$

T	100	$-\infty$	$-\infty$
100	99	98	$-\infty$

The recursive nature of values

- Flip the coin twice.



$$V^{\pi}(init) = 120 = 0.0 \cdot (0.5 \cdot (0 + 0.0 \cdot (0.5 \cdot 0 + 0.5 \cdot 100) + 1.0 \cdot (0.4 \cdot 0 + 0.6 \cdot 100)) + 0.5 \cdot (100 + 0.0 \cdot (0.5 \cdot 0 + 0.5 \cdot 100) + 1.0 \cdot (0.4 \cdot 0 + 0.6 \cdot 100)) \tag{11}$$

$$= 0.0 \cdot (0.5 \cdot (0 + 60) + 0.5 \cdot (100 + 60)) + 1.0 \cdot (0.4 \cdot (0 + 60) + 0.6 \cdot (100 + 60)) \tag{12}$$

$$= 0.0 \cdot (0.5 \cdot (0 + V(☺^1)) + 0.5 \cdot (100 + V(€^1))) + 1.0 \cdot (0.4 \cdot (0 + V(☺^1)) + 0.6 \cdot (100 + V(€^1))) \tag{13}$$

Bellman Equation (SuBa3.7)

$$
\begin{aligned}
V^\pi(s) &= \mathrm{E}_\pi\left\{R_t | s_t = s\right\} && (14)\\
&= \mathrm{E}_\pi\left\{\sum_{k=0}^{T}\gamma^k r_{t+k+1} | s_t = s\right\} && (15)\\
&= \mathrm{E}_\pi\left\{r_{t+1} + \gamma\sum_{k=0}^{T}\gamma^k r_{t+k+2} | s_t = s\right\} && (16)\\
&= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma\,\mathrm{E}_\pi\left\{\sum_{k=0}^{T}\gamma^k r_{t+k+2} | s_t = s\right\}\right] && (17)\\
&= \sum_a \pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right] && (18)
\end{aligned}
$$

- Values are recursively defined in terms of other values!

$$
\begin{aligned}
V^\pi(s^3) &= \sum_a \pi(s^3,a)\sum_{s'}\mathcal{P}^a_{s^3s'}\left[\mathcal{R}^a_{s^3s'} + \gamma V^\pi(s')\right] && (19)\\
&= \pi(s^3, LEFT)\sum_{s'}\mathcal{P}^{LEFT}_{s^3s'}\left[\mathcal{R}^{LEFT}_{s^3s'} + \gamma V^\pi(s')\right] && (20)\\
&= \pi(s^3, LEFT)\mathcal{P}^{LEFT}_{s^3s^2}\left[\mathcal{R}^{LEFT}_{s^3s^2} + \gamma V^\pi(s^2)\right] && (21)\\
&= 1.0\cdot 1.0\,[-1.0 + 1.0\cdot 99.0] && (22)\\
&= -1.0 + 99.0 && (23)
\end{aligned}
$$

s^0	s^1	s^2	$\boxed{s^3}$
s^4	s^5	s^6	s^7

$\pi(s)$

T	$<$	$<$	$\boxed{<}$
$\wedge$	$<$	$<$	$<$

$V^\pi(s)$

T	100	99	$\boxed{98}$
100	99	98	97

Outline

- Bellman Equation: a theoretical property of values
- Use it to make recursive algorithms to learn values
  - Policy Evaluation
  - Value Iteration

Policy Evaluation (SuBa4.1)

- Determine $V^\pi(s)$ for all states, given a certain policy π

Bellman equation "theory"

$$V^\pi(s) = E_\pi \left\{ \sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s \right\} \tag{24}$$

$$= E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \right\} \tag{25}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right] \tag{26}$$

Iterative Policy Evaluation "practice" (dynamic programming)

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \tag{27}$$

- $V_k(s)$ is k^{th} iteration
- $V_k(s)$ is an approximation of $V^\pi(s)$, with $V^\pi_{k=\infty}(s) = V^\pi(s)$

Iterative Policy Evaluation Algorithm (SuBa4.1)

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Figure : Iterative Policy Evaluation

- Δ: max difference (over all states) between previous value $V_{k-1}(s)$ and new value $V_k(s)$

Outline

Policy evaluation (SuBa4.1)

- Estimate $V^\pi(s)$ for a given policy π

π_0 (random policy)

T	?	?	?
?	?	?	?

π_0

Policy evaluation (SuBA4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

π_0 (random policy)

T	?	?	?
?	?	?	?

T	85	76	72
89	82	75	71

$V^{\pi_0}(s)$

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \tag{29}$$

Policy evaluation (SuBa4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

Policy Improvement (SuBa4.2)

- Idea: find an improved policy π', given estimated values $V^{\pi}(s)$

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V(s') \right] \tag{28}$$

π_0 (random policy)

T	?	?	?
?	?	?	?

$\pi_1(s) = \operatorname{argmax}_a \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi_0}(s') \right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

T	85	76	72
89	82	75	71

$V^{\pi_0}(s)$

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \xrightarrow{\text{impr.}} \pi_1 \tag{29}$$

Policy evaluation (SuBA4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

Policy Improvement (SuBA4.2)

- Idea: find an improved policy π', given estimated values $V^{\pi}(s)$

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right] \tag{28}$$

π_0 (random policy)

T	?	?	?
?	?	?	?

$\pi_1(s) = \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_0}(s')\right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

T	85	76	72
89	82	75	71

$V^{\pi_0}(s)$

T	100	99	98
100	99	98	97

$V^{\pi_1}(s)$

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \xrightarrow{\text{impr.}} \pi_1 \xrightarrow{\text{eval.}} V^{\pi_1} \tag{29}$$

Policy evaluation (SuBA4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

Policy Improvement (SuBA4.2)

- Idea: find an improved policy π', given estimated values $V^{\pi}(s)$

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right] \tag{28}$$

π_0 (random policy)

T	?	?	?
?	?	?	?

$\pi_1(s) = \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_0}(s') \right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

$\pi_2(s) = \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_1}(s') \right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

T	85	76	72
89	82	75	71

$V^{\pi_0}(s)$

T	100	99	98
100	99	98	97

$V^{\pi_1}(s)$

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \xrightarrow{\text{impr.}} \pi_1 \xrightarrow{\text{eval.}} V^{\pi_1} \xrightarrow{\text{impr.}} \ldots \pi_* \tag{29}$$

Policy evaluation (SuBa4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

Policy Improvement (SuBa4.2)

- Idea: find an improved policy π', given estimated values $V^{\pi}(s)$

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right] \tag{28}$$

π_0 (random policy)

T	?	?	?
?	?	?	?

T	85	76	72
89	82	75	71

$V^{\pi_0}(s)$

$\pi_1(s) = \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_0}(s')\right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

T	100	99	98
100	99	98	97

$V^{\pi_1}(s)$

$\pi_2(s) = \text{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_1}(s')\right]$

T	$<$	$<$	$<$
$\wedge$	$<$	$<$	$<$

T	100	99	98
100	99	98	97

$V^{\pi_2}(s)$

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \xrightarrow{\text{impr.}} \pi_1 \xrightarrow{\text{eval.}} V^{\pi_1} \xrightarrow{\text{impr.}} \dots \pi_* \xrightarrow{\text{eval.}} V^* \tag{29}$$

Policy evaluation (SuBA4.1)

- Estimate $V^{\pi}(s)$ for a given policy π

Policy Improvement (SuBA4.2)

- Idea: find an improved policy π', given estimated values $V^{\pi}(s)$

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V(s') \right] \tag{28}$$

Policy Iteration (SuBA4.3)

- Switch between evaluation and improvement

$$\pi_0 \xrightarrow{\text{eval.}} V^{\pi_0} \xrightarrow{\text{impr.}} \pi_1 \xrightarrow{\text{eval.}} V^{\pi_1} \xrightarrow{\text{impr.}} \ldots \pi_* \xrightarrow{\text{eval.}} V^* \tag{29}$$

- Converges to optimal values V^* and optimal policy π_* !

Value Iteration (next slide)

- Do policy improvement and policy iteration simultaneously

Value Iteration

Policy evaluation (SuBa4.1)

- Estimate $V^{\pi}(s)$ for a given policy π
- Update rule:

$$V_{k+1}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V_k(s') \right] \tag{30}$$

Value iteration (SuBa4.4)

- Informal idea: instead of summing over all actions, choose the action that leads to the state with the largest value
- Update rule:

$$V_{k+1}(s) = \max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V_k(s') \right] \tag{31}$$

Value Iteration Algorithm

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration

Policy Evaluation vs. Value Iteration

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
    $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Iterative Policy Evaluation vs. Value Iteration

Policy Evaluation vs. Value Iteration

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
$\quad \Delta \leftarrow 0$
$\quad$ For each $s \in \mathcal{S}$:
$\qquad v \leftarrow V(s)$
$\qquad V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
$\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
$\quad \Delta \leftarrow 0$
$\quad$ For each $s \in \mathcal{S}$:
$\qquad v \leftarrow V(s)$
$\qquad V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
$\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\quad \pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Iterative Policy Evaluation vs. Value Iteration

Policy Evaluation vs. Value Iteration

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Iterative Policy Evaluation vs. Value Iteration

Policy Evaluation vs. Value Iteration

Input π, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Iterative Policy Evaluation vs. Value Iteration

Prioritized Sweeping

- We have the basic algorithms to evaluate and improve policies
  - Policy evaluation: determine values, given a policy
  - Value iteration: determine values, improve policy along the way
- Now let's make them a bit more efficient!

Prioritized Sweeping

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

Prioritized Sweeping

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

- Updating all states at each iteration inefficient

Prioritized Sweeping

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

- Updating all states at each iteration inefficient
- Better: priority to update $V(s)$ if we think it will change
  - how do we know when it will change?
  - informally: “if a value changes, values of neighbouring states are likely to change also”

(neighbourhood relationship defined by $\mathcal{P}^a_{ss'}$)

Prioritized Sweeping

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

- Updating all states at each iteration inefficient
- Better: priority to update $V(s)$ if we think it will change
  - how do we know when it will change?
  - informally: “if a value changes, values of neighbouring states are likely to change also”

(neighbourhood relationship defined by $\mathcal{P}^a_{ss'}$)

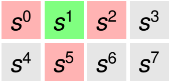
Prioritized Sweeping

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Figure : Value Iteration



- Updating all states at each iteration inefficient
- Better: priority to update $V(s)$ if we think it will change
  - how do we know when it will change?
  - informally: “if a value changes, values of neighbouring states are likely to change also”

(neighbourhood relationship defined by $\mathcal{P}^a_{ss'}$)

- Prioritized sweeping
  - prioritize updating states whose neighbours’ values have been updated

Prioritized Sweeping

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

V_k

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s') \right]$ (value update)

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

$\Delta(s) = \left| V_{k+1}(s) - V_k(s) \right|$ (value change)

s^0	s^1	s^2	s^3
s^4	s^5	s^6	s^7

$\forall s^- \quad C(s^-) = \sum_a \mathcal{P}^a_{s^- s} \Delta(s)$

s^1	s^3	s^6

priority queue – process state with highest value of $C(s)$ first.

http://eia.udg.es/~busquets/thesis/thesis_html/node52.html

Python

- This is not a Python course!
  - To make things easy, we provide a lot of “skeleton code”
  - Then you can focus on algorithmic aspects
  - Of course we will help with Python issues
- Coding perhaps a bit ‘scholarly’
  - If you are at ease with Python and RL, code whatever you like!
  - Next time: present list of projects
- Python version
  - our code is compatible with both Python2 and Python3
  - later we use PyBrain: only compatible with Python2
  - better to stick to Python2

Aims of Exercise 1

1. Think about MDPs
2. Understand the code in `MarkovDecisionProcess.py`
3. Implement policy evaluation (before next week)
4. Implement value iteration