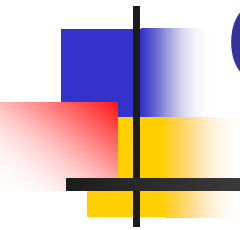
RN, Chapter 17
+ 21.2



Complex (Sequential) Decisions

Decision Theoretic Agents

- Introduction to Probability [Ch13]
- Belief networks [Ch14]
- Dynamic Belief Networks [Ch15]
- Single Decision [Ch16]
- **Sequential Decisions [Ch17]**
  - MDPs [Ch17.1-17.3]

    (Value Iteration, Policy Iteration)
  - POMDPs [Ch17.4-17.5]

    Dynamic Decision Networks
- Game Theory [Ch17.6 – 17.7]

Sequential Decisions

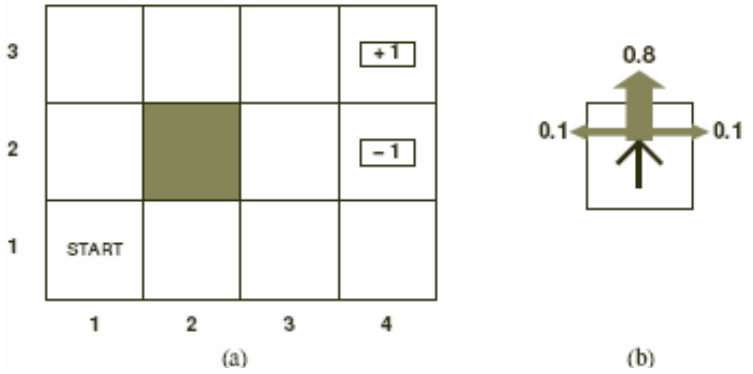
Environ/ Actions	Action(s)	
	Single	*Sequence*
Deterministic	Utility Problem	Planning
Stochastic	(Simple) Decision Problem	**Sequential Decision Problem**

- **Plan** ≈ sequence of
  DETERMINISTIC actions in
  DETERMINISTIC ENVIRONMENT
  But now: Actions/Environment stochastic
- **(Simple) Decision Problem** deals with SINGLE action
  (Reward after each action)
  But now: Sequence of actions

- State: { UnObservable / Accessible / InAccessible }

Environment + Effect of Actions: { Known / Unknown }

Sequential Decision Problem

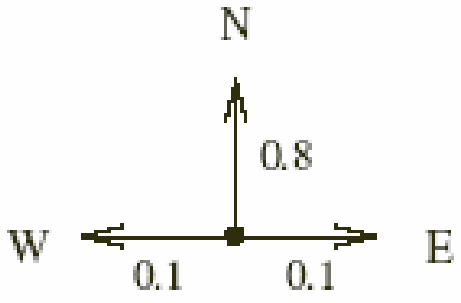


- Agent can move { North, South, East, West }
  Terminates on reaching [4, 2] or [4, 3]

- Transition Model: $\mathbf{M^a_{ij} = P(S_{t+1} = j \mid S_t = i, A = a)}$
  = prob of reaching state j if perform action a from state i

- If actions have DETERMINISTIC EFFECTS ($M^a_{ij} \in \{0, 1\}$)
  $\Rightarrow$ (std) Planning Problem

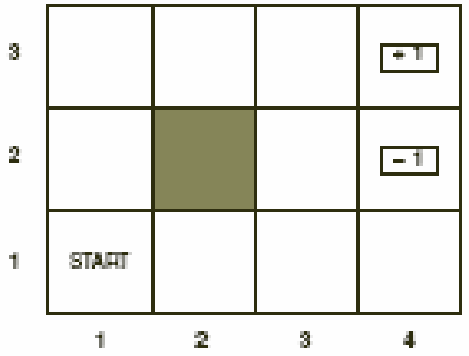
- but. . . Actions NOT reliable:
  Action = North $\equiv$



Immediate Reward & Total Utility

- Immediate reward:

$$R(S) = \begin{cases} -\frac{1}{25} & \text{if } S \neq [4,2] \text{ or } [4,3] \\ 1 - \frac{1}{25} & \text{if } S = [4,3] \\ -1 - \frac{1}{25} & \text{if } S = [4,2] \end{cases}$$



- Sequence of actions+states

$[s_0, a_1, s_1, \ldots, a_n, s_n]$

$$\sum_t R(s_t) = \begin{cases} 1 - \frac{n}{25} & \text{if } s_n = [4,3] \\ -1 - \frac{n}{25} & \text{if } s_n = [4,2] \end{cases}$$

- Utility depends on EPISODE

  Most reward after SERIES of states (≡ sequence of actions)

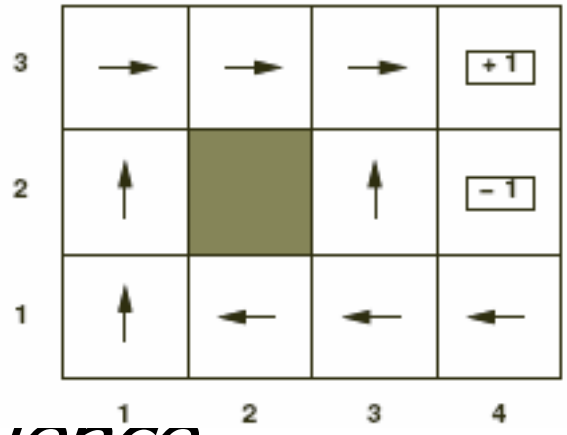
  … not single state

Observable (Accessible) Environments

- After each action,
  agent can observe resulting state S_{t+1}

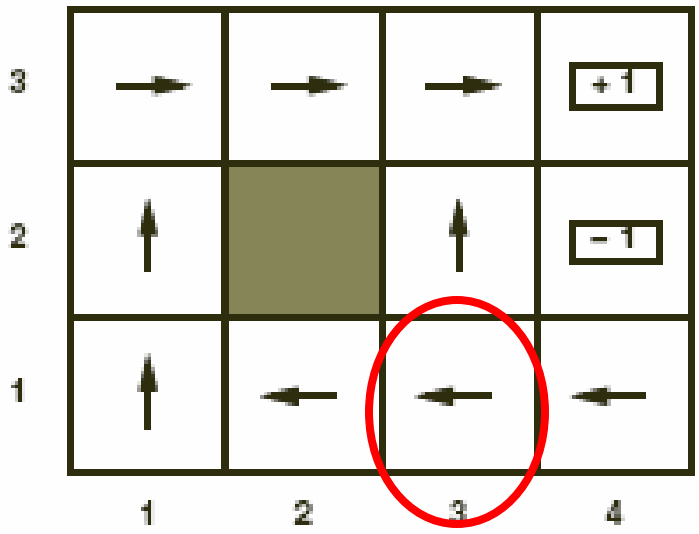
$\Rightarrow$ Agent just needs to know
  optimal action for each state

- Agent $\approx$ "Policy"
  π: State $\rightarrow$ Action

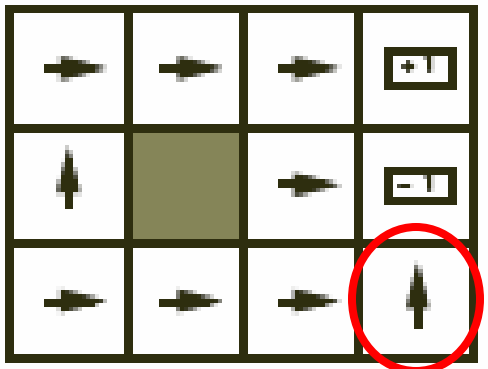


- Don't need *optimal action sequence*
  … just need optimal policy!
  - Optimize expected (cumulative) utility
- Agent, using π, is "deterministic" ("reflex")

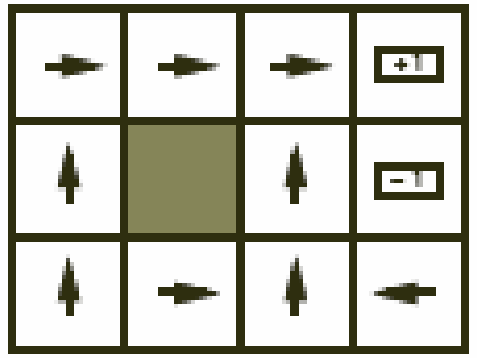
Different Rewards ⇒Different Policy



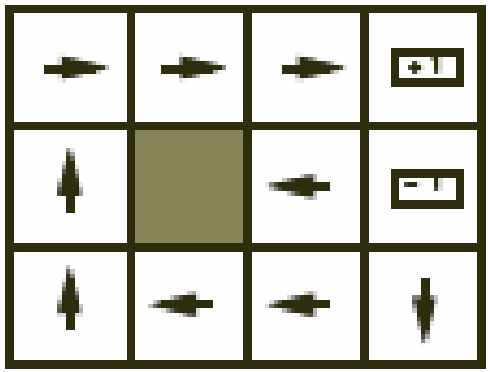
Better to go around, than risk "-1"



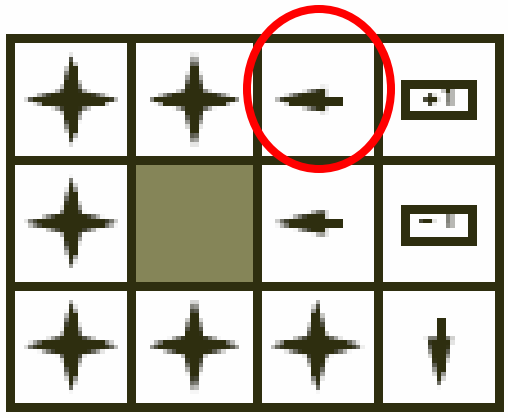
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$

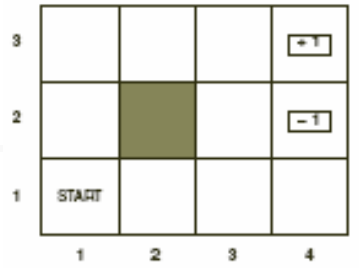


$-0.0218 < R(s) < 0$



$R(s) > 0$

Markov Decision Process



- Assume: { finite set of states S ; finite set of actions A

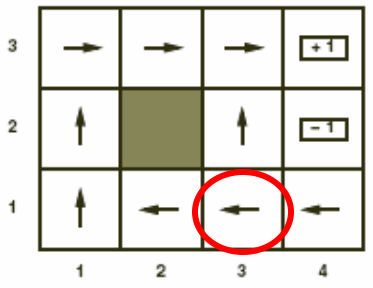
$\{ \uparrow \rightarrow \leftarrow \downarrow \}$

- At each discrete time, agent ...
  - observes state $s_t \in S$, and
  - chooses action $a_t \in A$

  then
  - receives (immediate) reward $r_t = R(s_t)$
  - state changes to $s_{t+1} \sim P(S_{t+1} \mid s_t, a_t)$
- **Markov assumption**:

  r_t, s_{t+1} depend only on current { state ; action

Defining Utility Fn Over States



- $U_\pi(S_i)$ = expected cumulative reward of executing policy π, starting in state S_i

  (aka "value function")

$$U_\pi([3,1]) = R([3,1]) + \sum_j P(S_j \mid [3,1] , \pi([3,1])) \times U_\pi(S_j)$$

$$= -0.4 + \sum_j P(S_j \mid [3,1] , \leftarrow) \times U_\pi(S_j)$$

$$= -0.4 + [0.8 \times U_\pi([2,1]) + 0.1 \times U_\pi([3,2]) + 0.1 \times U_\pi([3,1])]$$

Defining Utility Fn Over States

- $U_\pi(S_i)$ = expected cumulative reward of executing policy π, starting in state S_i

  (aka "value function")

  - $U_\pi(S_i) = R(S_i) + \sum_j P(S_j \mid S_i, \pi(\mathbf{S_i}))\ U_\pi(S_j)$

    $= R(S_i) + \sum_j M_{ij}^{\pi(\mathbf{S_i})}\ U_\pi(S_j)$

- **Theorem** (Bellman and Dreyfus) [MEU]

  Optimal policy $\pi^*(S)$ =

  action that maximizes expected utility $U^*(.)$

$$\pi^*(S_i) = \arg\max_a \sum_j M_{ij}^a U^*(S_j)$$

$$U^*(S_i) = R(S_i) + \max_a \sum_j M_{ij}^a U^*(S_j)$$

Fixed point

Finding Optimal Policy

- Easy to find optimal policy π^*, given U^*

$$\pi^*(S_i) = \text{argmax}_a \sum_j M^a_{ij} \mathbf{U^*}(S_j)$$

- Easy to find optimal utility U^*, given π^*

$$U^*(S_i) = R(S_i) + \sum_j M_{ij}^{\pi(S_i)} \times U^*(S_j)$$

- Circular:

  Use U^* to define π^*

  But need π^* to find U^* values

  [$U^*(S)$ depends on best action from S – ie, on $\pi^*(S)$]

- A: Use iterative algorithm...

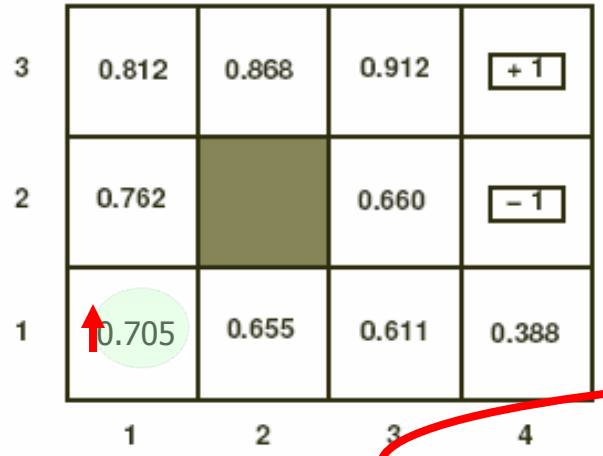
Value Iteration: Algorithm for computing U^*

function ValueIteration(…)
  $U_0(s) := R(s)$ for all s
  while $U(.)$ is changing do
    for each state S_i do
      $U_{t+1}(s_i) := R(s_i) + \max_a \sum_j M^a_{ij} \; U_t(s_j)$
    end *% for*
  end *% while*
  return $U_t(.)$

- $U(.)$ converges to stable values
- Each update of U is a *Bellman backup*

Example of Bellman Backup

Convergence! Bellman Backup

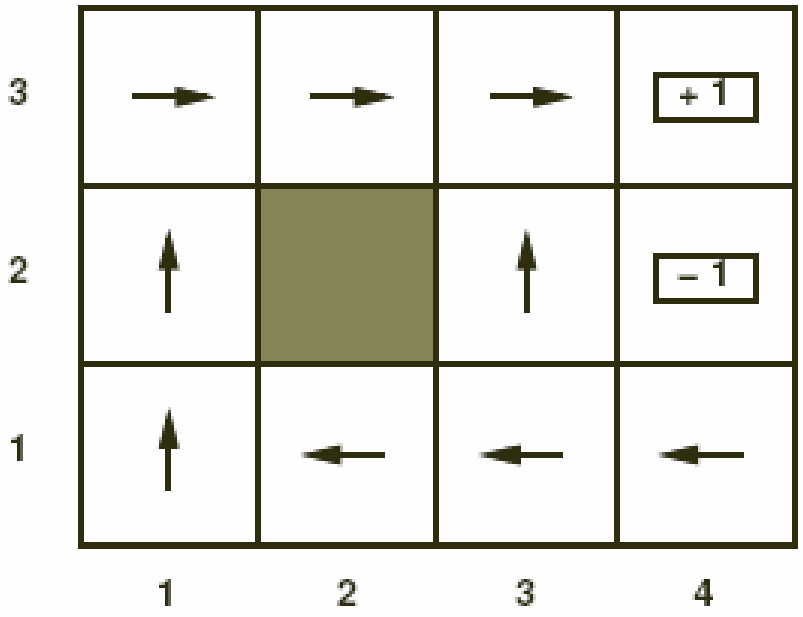


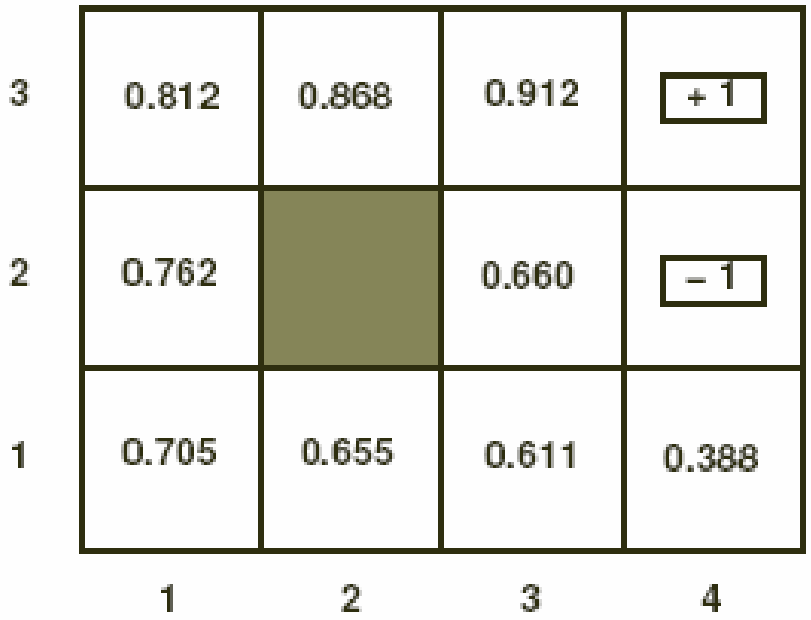
$U_{t+1}([1,1])$

0.709 -0.04
= 0.705

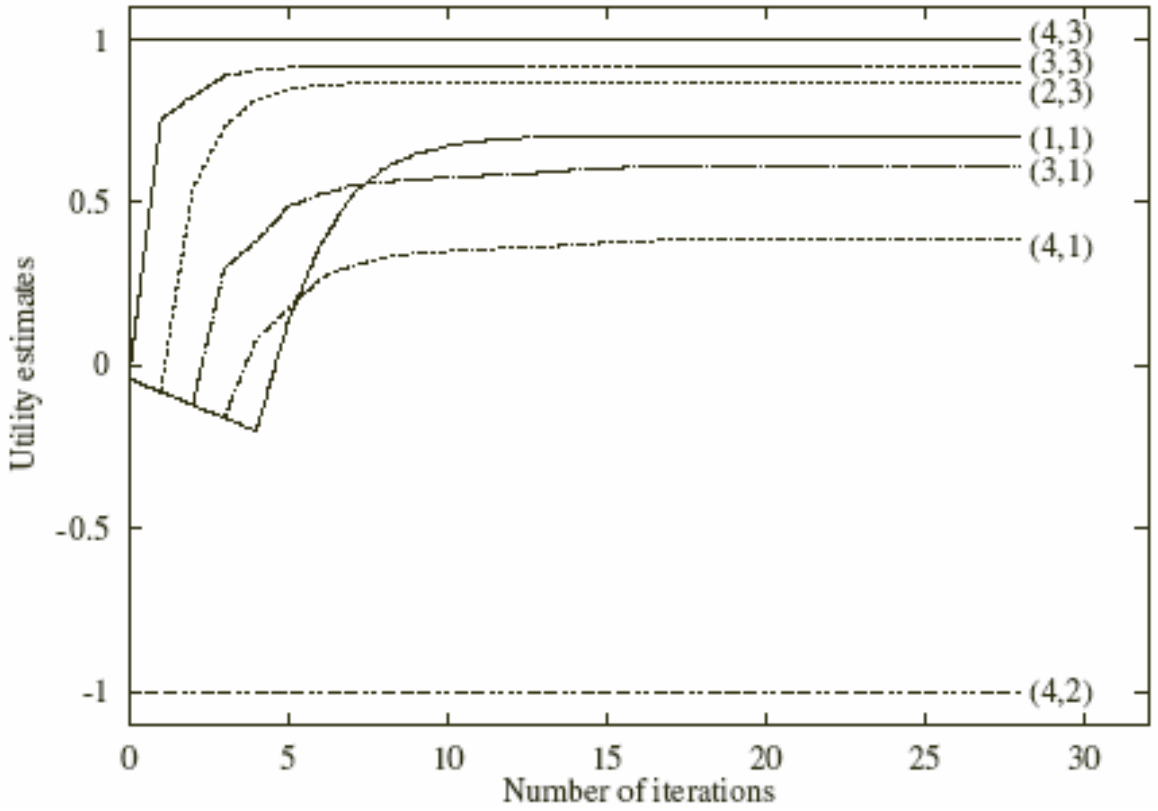
		S′	P(S′ | S, A)	$U_t(S')$	$E[U_t(S,A)]$
N	[1,1]	[1,1]	0.1	0.705	
		[1,2]	0.8	0.762	**0.709**
		[2,1]	0.1	0.655	
W	[1,1]	[1,1]	0.1	0.705	
		[1,1]	0.8	0.705	**0.708**
		[1,2]	0.1	0.762	
S	[1,1]	[1,1]	0.1	0.705	
		[1,1]	0.8	0.705	**0.700**
		[2,1]	0.1	0.655	
E	[1,1]	[1,1]	0.1	0.705	
		[2,1]	0.8	0. 655	**0.672**
		[1,2]	0.1	0.762	

Optimal Value Function & Policy





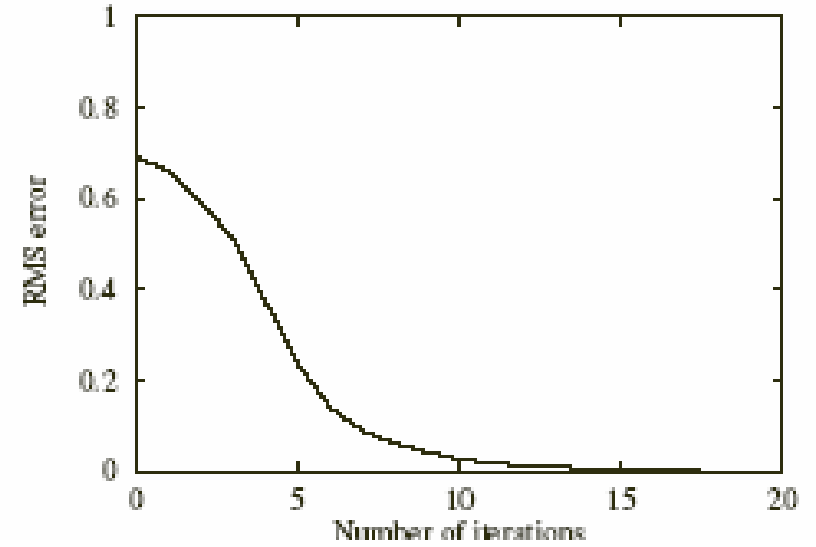
Behavior of Value Iteration

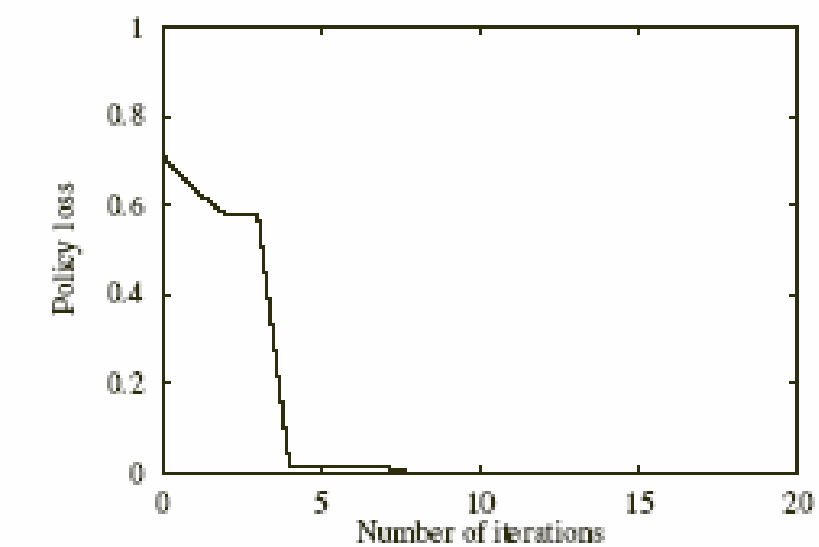


3	0.812	0.868	0.912	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Utility value for selected states…
at each iteration step in *ValueIteration*

Convergence of Value Iteration





$$RMS(\hat{U}) = \frac{1}{|S|}\sum_{S}\left(U^{*}(S) - \hat{U}(S)\right)^{2}$$

$$PolicyLoss(\hat{\pi}) = \sum_{S_j \in Terminal} U(S_j)\cdot[\, P(\pi^{*} \text{ leads } S_0 \text{ to } S_j) - P(\hat{\pi} \text{ leads } S_0 \text{ to } S_j)\,]$$

Convergence Rate

If use discount factor γ, then …

- Get ε-close after

  $N = \log(2 R_{max}/\varepsilon(1-\gamma)) / \log(1/\gamma))$

- If $|| U_{i+1} - U_i || < \varepsilon(1-\gamma)/\gamma$
  then $|| U_{i+1} - U^* || < \varepsilon$

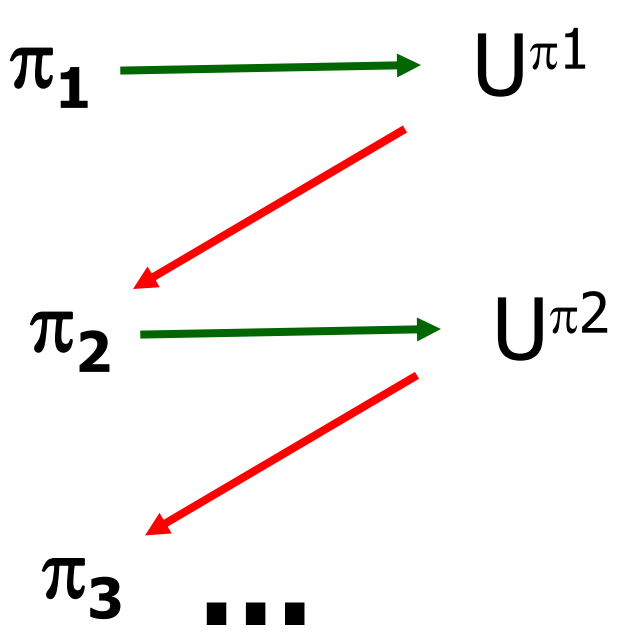
- Policy Loss:
  If $|| U_i - U^* || < \varepsilon$
  then $|| U^{\pi i} - U^* || < 2 \varepsilon \gamma /(1-\gamma)$

Policy Iteration

- Policy $\hat{\pi}$ not particularly sensitive to utility estimate $\tilde{U}$
- Why not just estimate Policy, on each iteration?

function *PolicyIteration*(M^a_{ij}, R(.))
  Choose initial policy π_0, t := 0
  Repeat
    % Value Determination
    Compute utility $U_{\pi_t}(s)$, $\forall$ s
    % Policy Improvement
    Compute new policy
      $\forall$ s_i: $\pi_{t+1}(s_i) := \text{argmax}_a \sum_j M^a_{ij} U_{\pi_t}(s_j)$
  until convergence ($\pi_{t+1} \approx \pi_t$)
  return(π_t)

Iterative 2-step Process



A. Value Determination
≡ Compute utility of current policy

$\forall\ s_i$: $U_t(s_i) := R(s_i) + \sum_j M^{\pi(si)}_{ij}\ U_t(s_j)$

B. Policy Improvement
≡ Compute new policy

$\forall\ s_i$: $\pi_{t+1}(s_i) := \text{argmax}_a \sum_j M^a_{ij}\ U_t(s_j)$

ValueDetermination

- **Thrm:** Policy iteration converges
  in fixed # of iterations
- PolicyImprovement is fast
  - $\forall\ s_i$: $\pi_{t+1}(s_i) := \text{argmax}_a \sum_j M^a_{ij}\ U_t(s_j)$
- Major expense ≡ "Evaluating Fixed Policy"
  - ValueDetermination
  - $\forall\ s_i$: $U_t(s_i) := R(s_i) + \sum_j M^{\pi(si)}_{ij}\ U_t(s_j)$
- Approaches

  (a) Linear Program

  (b) Adaptive dynamic programming

  (c) "Sampling" (Naïve updating)

  (d) Temporal Difference Learning
  "in between (b) and (c)"

(a) Linear Program

- Evaluating Fixed Policy $\equiv$
  Finding $\{ U(s) \}_s$ s.t.

$$U(s_i) = R(s_i) + \sum_j M_{ij}^{\pi(s_i)} U(s_j)$$

(in known accessible environment)

- Just solve set of equations:

$$\begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{pmatrix} + \begin{pmatrix} M_{11}^{\pi(s_1)} & M_{12}^{\pi(s_1)} & \dots & M_{1n}^{\pi(s_1)} \\ M_{21}^{\pi(s_2)} & M_{22}^{\pi(s_2)} & \dots & M_{2n}^{\pi(s_2)} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n1}^{\pi(s_n)} & M_{n2}^{\pi(s_n)} & \dots & M_{nn}^{\pi(s_n)} \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix}$$

$U = (I - M)^{-1} R$

(Note: most $M_{ij} = 0$)

but ... too many equations/unknowns!

- Eg, for backgammon:

  $\approx 10^{50}$ equations w/ 10^{50} unknowns! ... Unfeasibly large!

(b) Adaptive Dynamic Programming

- Guess initial values { $U(s_i)$ }
- Iteratively re-assign

$$U^{\pi}(s) := R(s) + \sum_{s'} P(s' \mid s, \pi(\mathbf{s})) \times U^{\pi}(s')$$

"simple backup"

- May need to examine each state s many times. . .

$\Rightarrow$ *AdaptiveDynamicProgramming ADP*

- … *ValueIteration* without "max":

function ADP(π, …)

$U^{\pi}_0(s) := R(s)$ for all s

while $U^{\pi}(.)$ is changing do

for each state s_i do

$U^{\pi}_{t+1}(s_i) := R(s_i) + \sum_j M^{a}_{ij} U^{\pi}_t(s_j)$

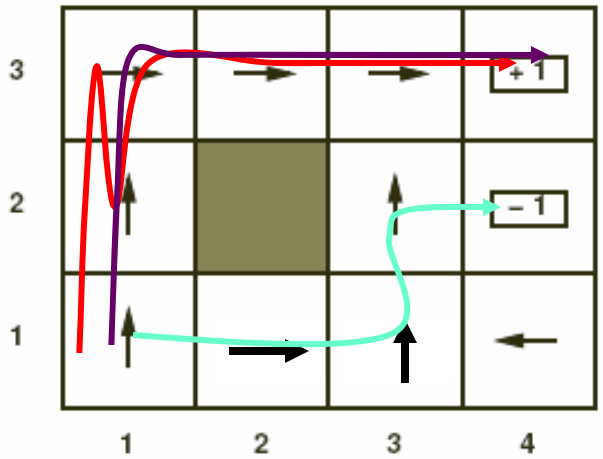
end *% for*

end *% while*

return $U^{\pi}_t(.)$

(c) "Sampling" (aka Naïve Updating)

Chapter 21.2



- Watch: Agent meandering thru MDP:

$[1,1]_{-.04} \rightarrow [1,2]_{-.04} \rightarrow [1,3]_{-.04} \rightarrow [1,2]_{-.04} \rightarrow [1,3]_{-.04} \rightarrow [2,3]_{-.04} \rightarrow [3,3]_{-.04} \rightarrow [4,3]_{+1}$

$[1,1]_{-.04} \rightarrow [1,2]_{-.04} \rightarrow [1,3]_{-.04} \rightarrow [2,3]_{-.04} \rightarrow [3,3]_{-.04} \rightarrow [4,3]_{+1}$

$[1,1]_{-.04} \rightarrow [2,1]_{-.04} \rightarrow [3,1]_{-.04} \rightarrow [3,2]_{-.04} \rightarrow [4,2]_{-1}$

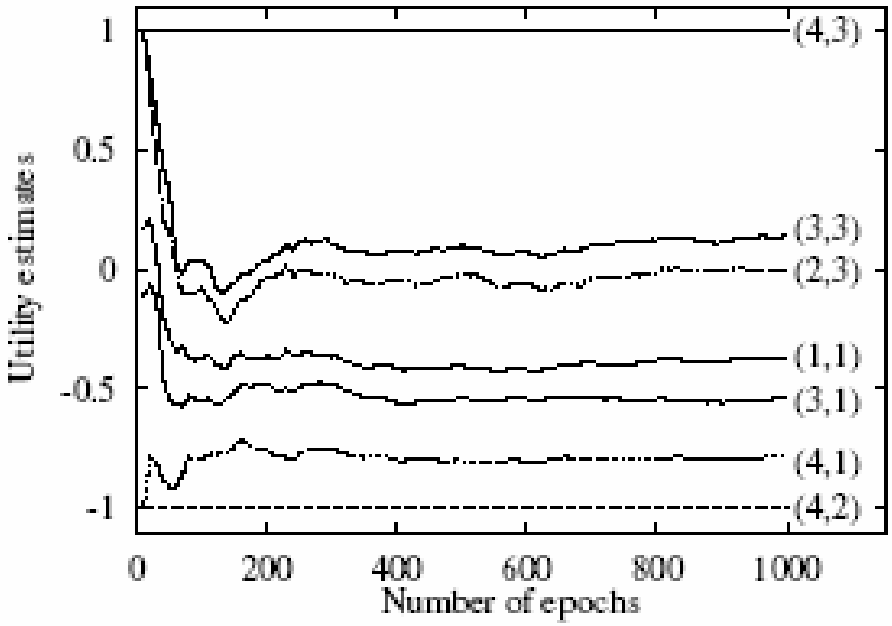
- Collect total score, starting from each state
  - from [1,1]: ⅓ × {[1– 7/25] + [1 – 5/25] + [-1 – 4/25]} = 0.12
  - from [1,2]: ⅓ × {[1 – 6/25] + [1 – 4/25] + [1 – 4/25]} = 0.813
  - from [3,3]: ½ × {[1 – 1/25] + [1 – 1/25]} = 0.96
  - …

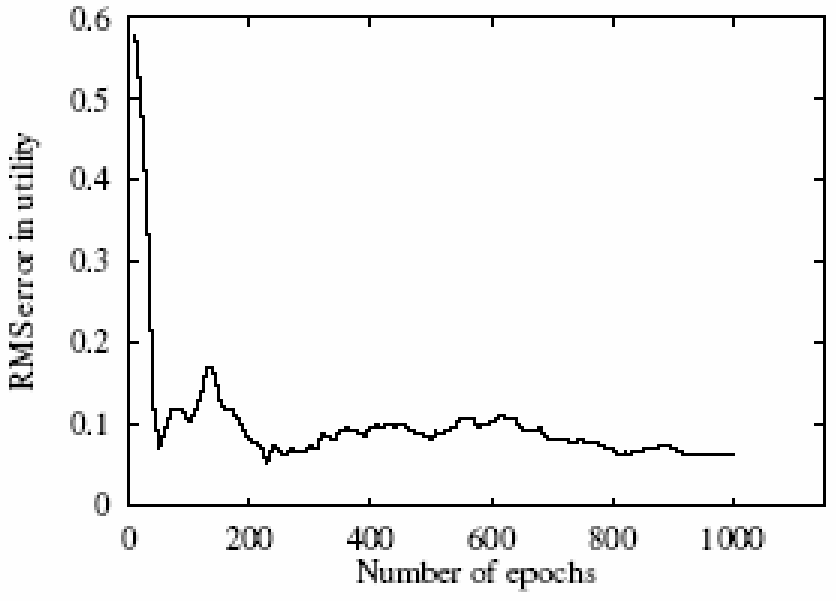
Sampling (con't)

- Simulate agent following policy
  For each state,
  - Collect statistics on utility
    - Eg, from [2, 3], how often reached +1 vs -1… in how many steps?
  - Compute average

  Provably converges to true *expected utilities*
- **Pro**: Only considers RELEVANT part of space
- **Con**: slow convergence
  - Even in small world, requires
    > 1000 sequences to get small error
    (< 0.1 root-mean-square error)

Convergence Graphs
Naïve Updating





Better Approach

- Must get to end before ANYTHING!

Q: Can we do better than sampling?

... by exploiting agent's knowledge of

- probability of state transitions
- final rewards
- ...

A: Yes!

- just recall Dynamic Programming
- + iteratively in Adaptive Dynamic Programming

- Now combine DP with Sampling…
  - Stochastic approximation to backups
    (to evaluate fixed policy)
  - Combine "sampling" with "calculation"
    … use sampling to solve set of equations. . .

(d) Temporal Difference Learning

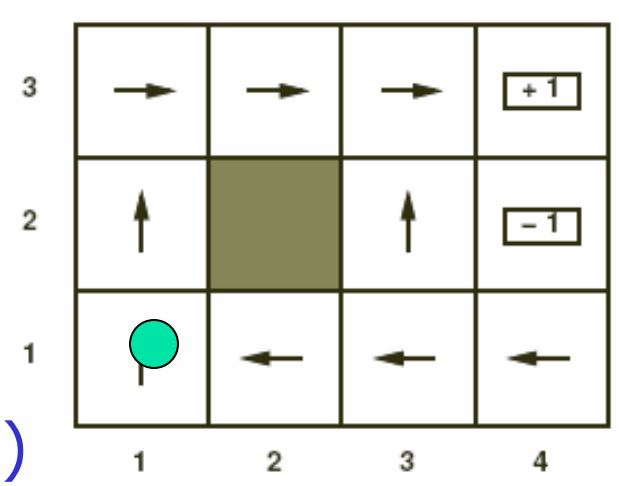
Assume π is deterministic

- Watch π-agent wandering around:

  At time *t*, agent at [1,1] with utility (estimate): $\tilde{U}_t([1,1]) = 0.5$

  Agent
  - receives reward $R([1,1]) = -0.04$
  - transitions to π: $[1,1] \rightarrow [1,2]$

    ... with utility (estimate): $\tilde{U}_t([1,2]) = 0.74$



- Recall

$$U([1,1]) = R([1,1]) + \sum_j M_{ij}^{\pi([1,1])} U(s_j)$$

$$= R([1,1]) + \sum_j M_{[1,1],j}^{\pi([1,1])} U(s_j)$$

$$= R([1,1]) + 1.0 \times U([1,2])$$

$$= R([1,1]) + U([1,2])$$

$$= -0.04 + ? \; U([1,2]) \; ?$$

So TWO estimates of $U([1,1])$: 0.5 and $-0.04 + U([1,2])$

(d) Temporal Difference Learning

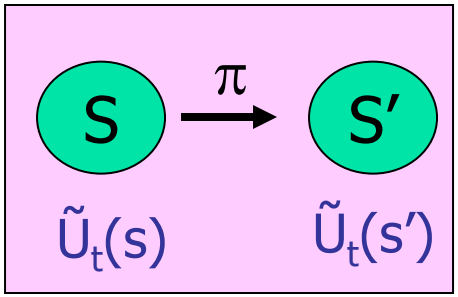
- Watch π-agent wandering around:

  At time t, agent at s with utility (estimate): $\tilde{U}_t(s)$

  Agent
  - receives reward $R(s)$
  - transitions to $s' := \pi(s)$

    ... with utility (estimate): $\tilde{U}_t(s')$

  $\Rightarrow$ Utility of s should be $\equiv R(s) + U(s')$



- Option1: Set $\tilde{U}_{t+1}(s) := R(s) + U(s')$
  - But $U(s')$ unknown!
- Option2: Set $\tilde{U}_{t+1}(s) := R(s) + \tilde{U}_t(s')$
  - But $\tilde{U}_t(s')$ might also be wrong!
- Option3: Move $\tilde{U}_{t+1}(s)$ to be "closer" to

  $R(s) + \tilde{U}_t(s')$

Adjusting $\tilde{U}_t$

- **Use observed transition to adjust utility of observed state,**

  to bring it closer to constraint equations
- When observing transition from s to s'

  bring $U(s)$ closer to $R(s)+U(s')$ using

  $\tilde{U}_{t+1}(s) := (1-\alpha)\,\tilde{U}_t(s) + \alpha(R(s)+\tilde{U}_t(s'))$
- **Temporal-difference (TD) equation**

  … difference between temporally successive states
- Justification:

  s' closer to end, so typically more accurate

Temporal-difference Equation

$$\tilde{U}_{t+1}(s) := (1 - \alpha)\, \tilde{U}_t(s) + \alpha(R(s) + \tilde{U}_t(s'))$$

- At equilibrium: $\tilde{U}(s) \equiv R(s) + \tilde{U}(s')$

  $\Rightarrow \tilde{U}_{t+1}(s) = \tilde{U}_t(s)$

  - α is learning rate... typically $\alpha \in [0{:}01, 0{:}5]$,
- If $M^{\pi(i)}_{ij}$ stochastic:
  - Transitions to $\{ s_1, \ldots, s_k \}$ with prob $\{ p_1, \ldots, p_k \}$:
  - Empirically, will consider each $R(s_i) + \tilde{U}_t(s_i)$ with prob p_i ... so perfect!

Why TD-Learning Works?

- Why is

  $U(s_i) := (1-\alpha)\, U(s_i) + \alpha\, [\, R(s_i)+U(s_j)\,]$ (**)

  effective at solving/approximating:

  $U(s_i) = R(s_i) + \sum_j M_{i,j}\, U(s_j)$ (*)
- If M_{ij} deterministic: OBVIOUS!

  (as only one neighboring state)
- If stochastic:

  (*) requires weighted average over neighboring states...
- No problem!

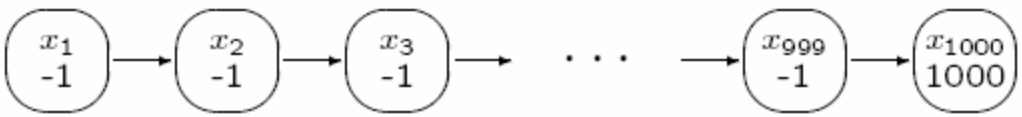
  Over time, agent samples from transitions out of *i*

  So, successive applications of (**)

  will average over neighboring states!

  (Need to keep α appropriately small)

Example of TD-Learning



- Immediate "Reward":

$$R(s_i) = -1 \quad \text{for } i = 1..999$$
$$R(s_{1000}) = 1000$$

- Policy: $\pi(x_i) = \text{Next}$
  so $x_i \rightarrow x_{i+1}$

  Transition Probability:

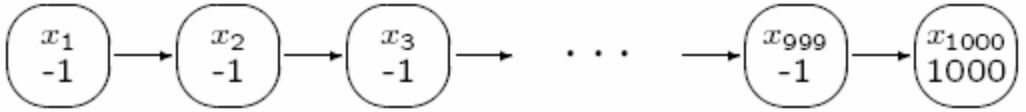
$$M_{ij}^N = \begin{cases} 1 & \text{if } j = i+1 \\ 0 & \text{otherwise} \end{cases}$$

- Utility:
  initially unknown (except $U(s_{1000}) = 1000$)

  For $i = 1..999$
  $U(s_i) = R(s_i) + U(s_{i+1})$

- . . . use $\alpha = 0.5$

How TD-Learning computes U(s)



- Initialize $U_0(s_i) = R(s_i)$ *(or whatever)*
- Sweep **#1**:

$U_1(s_1) := \alpha U_0(s_1) + (1-\alpha)[R(s_1)+U_0(s_2)]$
$= 0.5 \times -1 + 0.5[-1 + -1] = -1.5$

$U_1(s_2) := \alpha U_0(s_2) + (1-\alpha)[R(s_2)+U_0(s_3)]$
$= 0.5 \times -1 + 0.5[-1 + -1] = -1.5$

...

$U_1(s_{998}) := \alpha U_0(s_{998}) + (1-\alpha)[R(s_{998}) + U_0(s_9$
$= 0.5 \times -1 + 0.5[-1 + -1] = -1.5$

$U_1(s_{999}) := \alpha U_0(s_{999}) + (1-\alpha)[R(s_{999}) + U_0(s_1$

- Sweep **#2**:

$U_2(s_1) := \alpha U_1(s_1) + (1-\alpha)[R(s_1)+U_1(s_2)]$
$= 0.5 \times -1.5 + 0.5[-1 + -1.5] = -2$

$U_2(s_2) = -2 \quad \ldots \quad U_2(s_{997}) = -2$

$U_2(s_{998}) := \alpha U_1(s_{998}) + (1-\alpha)[R(s_{998}) + U_1(s_9$
$= 0.5 \times -1.5 + 0.5[-1 + 449] = 223$

$U_2(s_{999}) := \alpha U_1(s_{999}) + (1-\alpha)[R(s_{999}) + U_1(s_1$
$= 0.5 \times 449 + 0.5[-1 + 1000] = 724$

- Sweep **#3**:

For $i = 1..996$
$U_3(s_i) = -2.5$

$U_3(s_{997}) = 122.62$
$U_3(s_{998}) = 496.1$
$U_3(s_{999}) = 874.0$

- Sweep **#4**:

For $i = 1..995$
$U_4(s_i) = -3$

$U_4(s_{996}) = 59.6$
$U_4(s_{997}) = 309.9$
$U_4(s_{998}) = 685.6$
$U_4(s_{999}) = 936.5$

- Sweep **#5**: ...

Trace of TD

Lambda = 0, Alpha = 0.5, Length = 8, NumRounds = 16

0:	-1.50	-1.50	-1.50	-1.50	-1.50	-1.50	499.00	1000
1:	-2.00	-2.00	-2.00	-2.00	-2.00	223.00	724.00	1000
2:	-2.50	-2.50	-2.50	-2.50	122.62	498.12	874.00	1000
3:	-3.00	-3.00	-3.00	59.56	309.88	685.56	936.50	1000
4:	-3.50	-3.50	27.78	184.22	497.22	810.53	967.75	1000
5:	-4.00	11.64	105.50	340.22	653.38	888.64	983.38	1000
6:	3.32	58.07	222.36	496.30	770.51	935.51	991.19	1000
7:	30.20	139.71	358.83	632.90	852.51	962.85	995.09	1000
8:	84.46	248.77	495.37	742.21	907.18	978.47	997.05	1000
9:	166.11	371.57	618.29	824.19	942.32	987.26	998.02	1000
10:	268.34	494.43	720.74	882.76	964.29	992.14	998.51	1000
11:	380.88	607.08	801.25	923.02	977.72	994.83	998.76	1000
12:	493.48	703.67	861.64	949.87	985.77	996.29	998.88	1000
13:	598.07	782.15	905.25	967.32	990.53	997.08	998.94	1000
14:	689.61	843.20	935.79	978.43	993.31	997.51	998.97	1000
15:	765.91	888.99	956.61	985.37	994.91	997.74	998.98	1000

ADP vs TD

- **TD**: Each time, TD adjusts utility of OBSERVED state to agree with single OBSERVED successor
- **ADP** adjusts utility of ALL states (propagated) to agree with ALL successors, weighted by $\widehat{M}_{ij} \approx M_{ij}$
- $\Rightarrow$ TD $\approx$ approx to ADP!

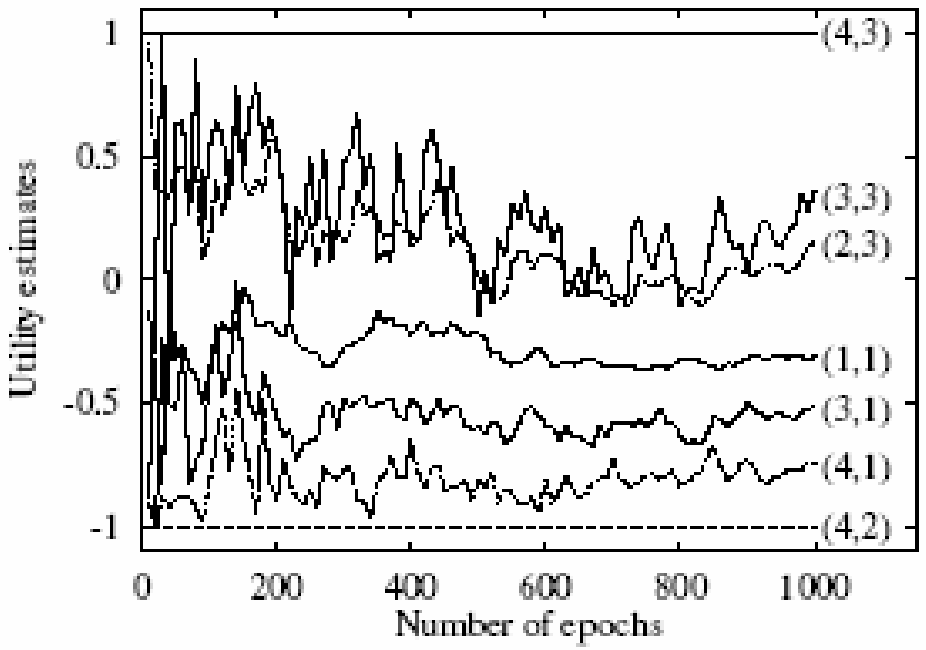
- "Prioritized sweeping"
  improve efficiency of ADP by adjusting just some utility values

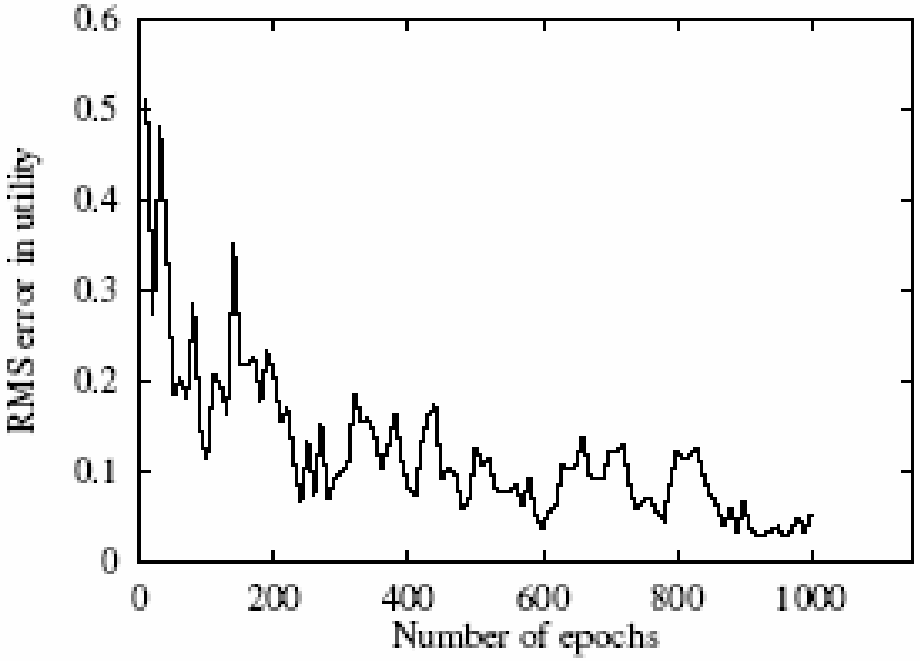
Performance

- Runs "noisier" than Naïve Updating
  but smaller error:
  In 4×3 world, RMS < 0.07 after 1000 examples!
- Deal w/ observed states during sample runs
  (not all instances,
  unlike Adaptive Dynamic Programming)

- In backgammon,
  considers only a few states,
  … only $\approx 10^5$ out of 10^{50}

Performance Curves



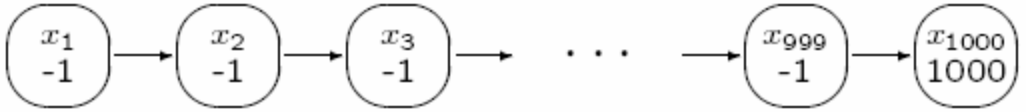


Limitation of TD-Learning…

Lambda = 0, Alpha = 0.5, Length = 8, NumRounds = 16

0:	-1.50	-1.50	-1.50	-1.50	-1.50	-1.50	499.00	1000
1:	-2.00	-2.00	-2.00	-2.00	-2.00	248.25	749.00	1000
2:	-2.50	-2.50	-2.50	-2.50	122.62	498.12	874.00	1000
3:	-3.00	-3.00	-3.00	59.56	309.88	685.56	936.50	1000
4:	-3.50	-3.50	27.78	184.22	497.22	810.53	967.75	1000
5:	-4.00	11.64	105.50	340.22	653.38	888.64	983.38	1000
6:	3.32	58.07	222.36	496.30	770.51	935.51	991.19	1000
7:	30.20	139.71	358.83	632.90	852.51	962.85	995.09	1000
8:	84.46	248.77	495.37	742.21	907.18	978.47	997.05	1000
9:	166.11	371.57	618.29	824.19	942.32	987.26	998.02	1000
10:	268.34	494.43	720.74	882.76	964.29	992.14	998.51	1000
11:	380.88	607.08	801.25	923.02	977.72	994.83	998.76	1000
12:	493.48	703.67	861.64	949.87	985.77	996.29	998.88	1000
13:	598.07	782.15	905.25	967.32	990.53	997.08	998.94	1000
14:	689.61	843.20	935.79	978.43	993.31	997.51	998.97	1000
15:	765.91	888.99	956.61	985.37	994.91	997.74	998.98	1000

Limitation of TD-Learning



- On Sweep#1:
  - TD-learning does NOTHING for i = 1..998
  - . . . helps $U(s_{999})$ as $U(s_{1000})$ meaningful
- On Sweep#2:
  - TD-learning does NOTHING for i = 1..997
  - . . . helps $U(s_{998})$ as $U(s_{999})$ less meaningless
  - . . . helps $U(s_{999})$ as $U(s_{1000})$ meaningful
- On Sweep#k:
  - TD-learning does NOTHING for i = 1..1000 –k
  - . . . helps $U(s_j)$ as $U(s_{j+1})$ less meaningless ($j>1000-k$)
- Takes 1000 sweeps to do ANYTHING for s_1 !

Propagation of Values... ⇒ Eligibility Trace

- Q: Why is TD-learning so slow?
- A: Only uses $\tilde{U}_t(s_{i+1})$ to help adjust $\tilde{U}_t(s_i)$
- So "factual" information takes many iterations to "percolate" back
- Q: Why not propagate back FARTHER?
- General form:

$$\tilde{U}_{t+1}(s) = (1-\alpha)\,\tilde{U}_t(s) + \alpha\, U^*(s)$$

where $U^*(s)$ is "better estimate of $U(s)$"

What is good estimate of U(s)?

- **Now**: just "look ahead **1**":

  $U^{(1)}(s_i) := r_i + \tilde{U}_n(s_{i+1})$

- Why not: Look ahead **2**:

  $U^{(2)}(s_i) := r_i + r_{i+1} + \tilde{U}_n(s_{i+2})$

  Look ahead **3**:

  $U^{(3)}(s_i) := r_i + r_{i+1} + r_{i+2} + \tilde{U}_n(s_{i+3})$

  ...

- Look ahead **k**:

  $U^{(k)}(s_i) = \sum_{t=0}^{k-1} r_{i+t} + \tilde{U}_n(s_{i+k})$

- **Or better**: Why not ALL of these estimates?

  $U(s) = (1-\lambda)\,[U^{(1)}(s) + \lambda\, U^{(2)}(s) + \lambda^2\, U^{(3)}(s) + \cdots]$

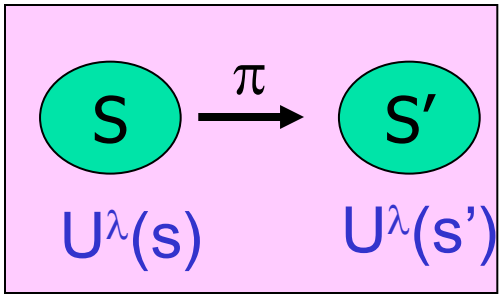
  for some $\lambda \in [0,1]$

What is good estimate of U(s)?

- **Or better**: Why not ALL of these estimates?

$$U^{\lambda}(s) = (1-\lambda)\,[U^{(1)}(s) + \lambda\, U^{(2)}(s) + \lambda^2\, U^{(3)}(s) + \cdots]$$

for some $\lambda \in [0,1]$



- **Implementation of** TD(λ):

$$U^{\lambda}(s') = R(s') + [\,(1-\lambda)\,\tilde{U}_t(s) + \lambda\, U^{\lambda}(s)\,]$$

$$\tilde{U}_{t+1}(s) = (1-\alpha)\,\tilde{U}_t(s) + \alpha\, U^{\lambda}(s)$$

- $\lambda = 0 \quad \Rightarrow \quad U^{\lambda}(s) = U^{(1)}(s)$

As $\lambda \rightarrow 1 \quad \Rightarrow \quad U^{\lambda}(s) \rightarrow U^{(k)}(s)$ for large k...

Computing $U^{\lambda}(s)$

- Consider trajectory $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$
- Suppose we have

$$U^{\lambda}(s_3) = (1-\lambda) \times [\, U^{(1)}(s_3) + \lambda\, U^{(2)}(s_3) + \lambda^2\, U^{(3)}(s_3) + \cdots \,]$$
$$= (1-\lambda) \times [\, (r_3 + \tilde{U}(s_2)) + \lambda(r_3 + r_2 + \tilde{U}(s_1)) + \lambda^2(r_3 + r_2 + r_1 + \tilde{U}(s_0)) + \cdots \,]$$

- Now reach s_4 … need to compute $U^{\lambda}(s_4)$

$$U^{\lambda}(s_4) = (1-\lambda) \times [\, U^{(1)}(s_4) + \lambda\, U^{(2)}(s_4) + \lambda^2\, U^{(3)}(s_4) + \cdots \,]$$
$$= (1-\lambda) \times [\, (r_4 + \tilde{U}(s_3)) + \lambda(r_4 + r_3 + \tilde{U}(s_2)) + \lambda^2(r_4 + r_3 + r_2 + \tilde{U}(s_1)) + \lambda^3(r_4 + r_3 + r_2 + r_1 + \tilde{U}(s_0)) + \cdots \,]$$
$$= (1-\lambda) \times [\, \tilde{U}(s_3) + r_4(1 + \lambda + \lambda^2 + \lambda^3 + \ldots) + \lambda\, [\, (r_3 + \tilde{U}(s_2)) + \lambda(r_3 + r_2 + \tilde{U}(s_1)) + \lambda^2(r_3 + r_2 + r_1 + \tilde{U}(s_0)) + \cdots \,]$$
$$= (1-\lambda) \times [\, \tilde{U}(s_3) + r_4 / (1-\lambda) + \lambda\, U^{\lambda}(s_3) / (1-\lambda) \,]$$
$$= r_4 + \lambda\, U^{\lambda}(s_3) + (1-\lambda)\, \tilde{U}(s_3)$$

Data – TD(λ)

Lambda = 0.3, Alpha = 0.5, Length = 8, NumRounds = 16

[0] UEst:	-1.35	-0.50	2.34	11.80	43.35	148.50	499.00	1000
ULambda:	-1.70	0.00	5.68	24.61	87.70	298.00	999.00	1000
[1] UEst:	0.67	6.50	22.59	65.18	170.35	398.25	749.00	1000
ULambda:	2.70	13.49	42.83	118.55	297.35	648.00	999.00	1000
[2] UEst:	10.06	29.75	74.95	170.41	347.51	610.62	874.00	1000
ULambda:	19.45	53.01	127.32	275.65	524.67	823.00	999.00	1000
[3] UEst:	34.09	78.70	164.44	311.27	523.55	760.56	936.50	1000
ULambda:	58.12	127.64	253.93	452.13	699.59	910.50	999.00	1000
[4] UEst:	79.21	156.42	282.28	461.03	670.61	857.41	967.75	1000
ULambda:	124.33	234.15	400.13	610.79	817.67	954.25	999.00	1000
[5] UEst:	147.76	257.89	412.41	598.53	781.32	916.77	983.38	1000
ULambda:	216.30	359.36	542.53	736.03	892.02	976.12	999.00	1000
[6] UEst:	236.77	372.70	539.24	712.75	859.09	951.91	991.19	1000
ULambda:	325.77	487.50	666.07	826.98	936.85	987.06	999.00	1000
[7] UEst:	339.21	489.29	651.98	801.02	911.09	972.22	995.09	1000
ULambda:	441.65	605.88	764.71	889.29	963.10	992.53	999.00	1000
[8] UEst:	446.41	598.15	745.38	865.61	944.61	983.74	997.05	1000
ULambda:	553.61	707.02	838.78	930.21	978.14	995.27	999.00	1000
[9] UEst:	550.30	693.23	818.58	910.91	965.61	990.19	998.02	1000
ULambda:	654.20	788.30	891.79	956.21	986.61	996.63	999.00	1000
[10] UEst:	644.86	771.87	873.46	941.62	978.47	993.75	998.51	1000
ULambda:	739.41	850.51	928.34	972.33	991.33	997.32	999.00	1000
[11] UEst:	726.52	834.06	913.11	961.86	986.20	995.71	998.76	1000
ULambda:	808.18	896.25	952.77	982.11	993.92	997.66	999.00	1000
[12] UEst:	794.00	881.42	940.90	974.90	990.77	996.77	998.88	1000
ULambda:	861.48	928.79	968.69	987.94	995.34	997.83	999.00	1000
[13] UEst:	847.69	916.35	959.87	983.14	993.44	997.34	998.94	1000
ULambda:	901.38	951.28	978.84	991.37	996.11	997.91	999.00	1000
[14] UEst:	889.04	941.41	972.54	988.25	994.98	997.65	998.97	1000
ULambda:	930.39	966.47	985.21	993.37	996.53	997.96	999.00	1000
[15] UEst:	919.99	958.96	980.83	991.38	995.87	997.81	998.98	1000
ULambda:	950.94	976.52	989.13	994.51	996.75	997.98	999.00	1000

Comparison – TD(0) vs TD(0.3)

Lambda = 0, Alpha = 0.5, Length = 8, NumRounds = 16

0:	-1.50	-1.50	-1.50	-1.50	-1.50	-1.50	499.00	1000
1:	-2.00	-2.00	-2.00	-2.00	-2.00	248.25	749.00	1000
2:	-2.50	-2.50	-2.50	-2.50	122.62	498.12	874.00	1000
3:	-3.00	-3.00	-3.00	59.56	309.88	685.56	936.50	1000
4:	-3.50	-3.50	27.78	184.22	497.22	810.53	967.75	1000
5:	-4.00	11.64	105.50	340.22	653.38	888.64	983.38	1000
6:	3.32	58.07	222.36	496.30	770.51	935.51	991.19	1000
7:	30.20	139.71	358.83	632.90	852.51	962.85	995.09	1000
8:	94.46	248.77	495.37	742.21	907.18	978.47	997.05	1000
9:	166.11	371.57	618.29	824.19	942.32	987.26	998.02	1000
10:	268.34	494.43	720.74	882.76	964.29	992.14	998.51	1000
11:	380.88	607.08	801.25	923.02	977.72	994.83	998.76	1000
12:	493.48	703.67	861.64	949.87	985.77	996.29	998.88	1000
13:	598.07	782.15	905.25	967.32	990.53	997.08	998.94	1000
14:	689.61	843.20	935.79	978.43	993.31	997.51	998.97	1000
15:	765.91	888.99	956.61	985.37	994.91	997.74	998.98	1000

Lambda = 0.3, Alpha = 0.5, Length = 8, NumRounds = 16

0:	-1.35	-0.50	2.34	11.80	43.35	148.50	499.00	1000
1:	0.67	6.50	22.59	65.18	170.35	398.25	749.00	1000
2:	10.06	29.75	74.95	170.41	347.51	610.62	874.00	1000
3:	34.09	78.70	164.44	311.27	523.55	760.56	936.50	1000
4:	79.21	156.42	282.28	461.03	670.61	857.41	967.75	1000
5:	147.76	257.89	412.41	598.53	781.32	916.77	983.38	1000
6:	236.77	372.70	539.24	712.75	859.09	951.91	991.19	1000
7:	339.21	489.29	651.98	801.02	911.09	972.22	995.09	1000
8:	446.41	598.15	745.38	865.61	944.61	983.74	997.05	1000
9:	550.30	693.23	818.58	910.91	965.61	990.19	998.02	1000
10:	644.86	771.87	873.46	941.62	978.47	993.75	998.51	1000
11:	726.52	834.06	913.11	961.86	986.20	995.71	998.76	1000
12:	794.00	881.42	940.90	974.90	990.77	996.77	998.88	1000
13:	847.69	916.35	959.87	983.14	993.44	997.34	998.94	1000
14:	889.04	941.41	972.54	988.25	994.98	997.65	998.97	1000
15:	919.99	958.96	980.83	991.38	995.87	997.81	998.98	1000

Dealing with Large Spaces

- Typical spaces $10^{20} - 10^{50}$

  How to represent U(s)?

A1: As explicit table?

- Need to *see* each state/action (often)
- Need to *store* each state/action

A2: Implicitly, as $U_\pi(s, W)$

- w/ adjustable param's W (eg, weights in NN)

- Eg: TD-Gammon:
  - input nodes encode board position
  - activation of output node gives utility
- To get "best" W, use temporal difference error

Use *temporal difference error*

$$J(W) = \frac{1}{2}\left(U_\pi(s, W) - [R(s) + \gamma U_\pi(s', W)]\right)^2$$

Gradient descent, using derivative (wrt 1^{st} W):

$$W \mathrel{-}= \alpha \nabla_W U_\pi(s, W)\ \left[U_\pi(s, W) - [R(s) + \gamma U_\pi(s', W)]\right]$$

Eligibility Trace

Now: Update W based only on $s_t \stackrel{a}{\mapsto} s_{t+1}$
Why not also use $s_1, s_2, \ldots, s_{t-1}$?

- Re-use $\nabla_{W_{t-i}} U(s_{t-i}, W_{t-i})$ for each s_{t-i}

  Update $U(s_t, W_t)$ with step $-\nabla_{W_t} U(s_t, W_t)$
  +
  small step $-\nabla_{W_{t-1}} U(s_{t-1}, W_{t-1})$,
  smaller step $-\nabla_{W_{t-2}} U(s_{t-2}, W_{t-2})$,
  ...

$$W \mathrel{-}= \alpha \left(\sum_{i=0}^{\infty} \lambda^i \nabla_{W_{t-i}} U_\pi(s_{t-i}, W_{t-i}) \right) \left(U_\pi(s, W) - [R(s) + \gamma U_\pi(s', W)] \right)$$

- Implementation:

  Maintain *current gradient vector*

  $$G_t := \lambda G_{t-1} + \nabla_{W_t} U_\pi(s_t, W_t)$$

- kinda like *momentum* but ...
  momentum: re-uses WEIGHT CHANGES
  eligibility: re-uses GRADIENT VECTORS

TD(λ) Alg: Computing value of policy

procedure **TD**(π, λ, α, s_0)

initialize $G := 0$; W randomly

s_0 is starting state

while W has not converged do

take action $a := \pi(s_t)$

observe resulting state s_{t+1} , reward $R(s_t)$

$G := \lambda G + \nabla_W U_\pi(s_t, W)$

$W \mathrel{-}= \alpha G(U_\pi(s_t, W) - [R(s_t) + \gamma U_\pi(s_{t+1}, W)]$

end while

return W *% which defines $U( )$ (eg, $U_\pi(s_0)$)*

end **TD**

Comments on TD(λ)

- TD(λ) computes value of fixed policy w/out direct access to
  - transition prob M_a^{ij}
  - immediate reward R(...)
- Use it to find better policy using policy iteration

  $\pi_{t+1}(s_i) := \text{argmax}_a \sum_j M^a_{ij} U_{\pi_t}(s_j)$

- Formal analyses

$$\|U - U_\pi(\cdot)\|_D = \left(\sum_s |U(s) - U_\pi(s)|^2 D(s)\right)^{1/2}$$

If . . .

$$\|U - U_\pi(\cdot)\|_D \leq \frac{\|U - U^*\|_D}{1 - \gamma(1-\lambda)} / (1 - \lambda\gamma)$$

The Curse of Dimensionality

- Computational cost scales with
  number of states, |S|
- |S| is exponential in
  number of "dimensions" (sensors)
- "Curse of Dimensionality" [Bellman]
  - **Value Iteration**: O(n |S| |A| B)
  - **Value Determination**: O(n' |S|B)
    or O($|S|^3$)
  - **Policy Iteration**: O(n" |S| |A|)

Simple Extensions

- So far,
  - ~~$R(s)$~~ → $R(s_{t+1} \mid s_t, a_t)$

    reward just depends on current state and action
  - ~~$P(s_{t+1} \mid s_t, a_t)$~~ → $R^q(s_{t+1} \mid s_t, a_t)$

    transition probability depends on : WHEN
  - ~~$\sum_{s_t} R_{s_t}$~~ → $\sum_{s_t} \gamma^t R_{st}$

    total cost discounted sum of rewards

Summary

- Sequential Decision Making
  - If Observable, Markovian (+ Stationary)
    $\Rightarrow$ Markov Decision Policy
  - Solution $\equiv$ **Policy** (map state to action)
  - "**Utility**" = value of policy ... dynamic programming
- Finding Policy:
  - Value Iteration (Bellman Backup)
  - Policy Iteration ... ValueDetermination
    - (Adaptive) Dynamic Programming
    - Naïve Sampling
    - TD(λ)

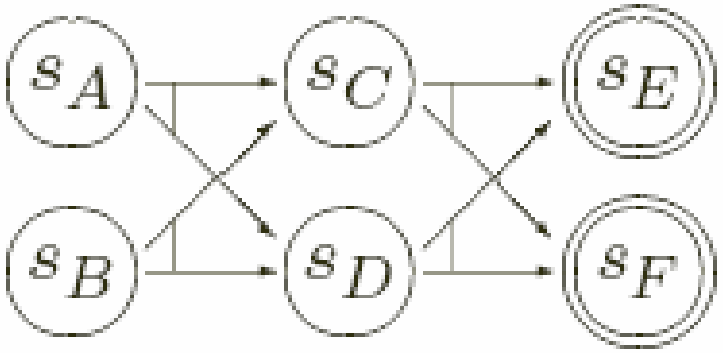
EXTRA:

Using Dynamic Programming

- Simple Task: **Value Determination**

  Given fixed policy π, compute $U^{\pi}(s)$

  (Expected value of starting at s, and following π)



- Only one action / non-terminal node:

  $M_{A,C} = 0.2$ $M_{A,D} = 0.8$

  $M_{B,C} = 0.4$ $M_{A,D} = 0.6$

  $M_{C,E} = 0.3$ $M_{C,E} = 0.7$

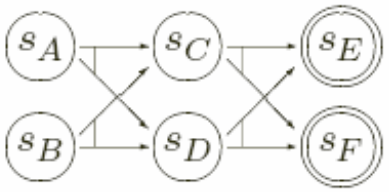
  $M_{D,E} = 0.1$ $M_{D,F} = 0.9$

- Rewards:

  $R(s_A) = R(s_B) = R(s_C) = R(s_D) = 0$

  $R(s_E) = -1$ $R(s_F) = +1$

Dynamic Programming ≡ Working Backwards



- As state @ time=3 is TERMINAL, only options { s_E, s_F }... known values:

Time =	1	2	3
s_A			
s_B			
s_C			
s_D			
s_E			$U^{\pi}(s_E) = R(s_E) = -1$
s_F			$U^{\pi}(s_F) = R(s_F) = +1$

Q: What is value of being in state s_D @ time=2?

A: Reward for being in s_D, PLUS value obtained by taking action $\pi(s_D)$

Dynamic Programming, II

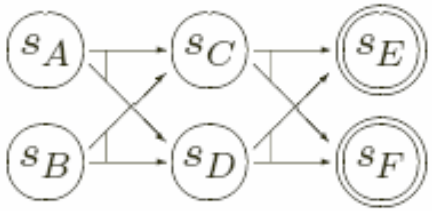
- $U^{\pi}(s_E) = R(s_E) + \sum_{s'} P(\, s' \mid s_C, \pi(s_C)\,) \times U^{\pi}(s')$
- Only non-zero values:

  $\{\, P(\, s_E \mid s_C, \pi(s_C)\,) = 0.3, P(\, s_F \mid s_C, \pi(s_C)\,) = 0.7 \,\}$
- N.b. already know
  - $U(\, s_E\,) = R(\, s_E\,) = -1$
  - $U(\, s_F\,) = R(\, s_F\,) = +1$

$\Rightarrow U(s_C) = 0 + [0.3 \times -1 + 0.7 \times +1] = 0.4$

- So. . .

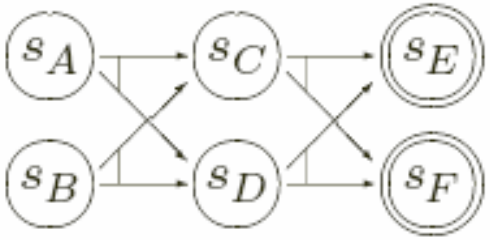
Time =	1	2	3
s_A			
s_B			
s_C		0.4	
s_D		0.8	
s_E			-1
s_F			+1



Dynamic Programming, III

- Q: What is value of being in state s_A @ time=1?
- A: Reward for being in s_A, PLUS value obtained by taking action $\pi(s_A)$.
- $U^{\pi}(s_A) = R(s_A) + \sum_{s'} P(\, s' \mid s_A, \pi(s_A)\,) \times U^{\pi}(s')$
  $= 0 + [M_{A,C}^{\pi(S_A)} \times U^{\pi}(S_C) + M_{A,B}^{\pi(S_A)} \times U^{\pi}(S_B)$
  $= 0 + [0.2 \times 0.4 + 0.8 \times 0.8] = 0.72$

Time =	1	2	3
s_A	0.72		
s_B	0.64		
s_C		0.4	
s_D		0.8	
s_E			-1
s_F			+1



Value Determination

- If *n* states, and *k* times,
  just need to do $n \times k$ computations...each . . .
  $U^{\pi}(s) = R(s) + \sum_{s'} P(\, s' \mid s, \pi(s)\,) \times U^{\pi}(s')$

Q1: What if many actions?

- A: Fixed policy; so single action $\pi(s)$ for s

Q2: What if $\exists$ loops? What before what...

($S_A \rightarrow \ldots \rightarrow S_A$)

$$\begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{pmatrix} + \begin{pmatrix} M_{11} & M_{21} & \ldots & M_{n1} \\ M_{12} & M_{22} & \ldots & M_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ M_{1n} & M_{2n} & \ldots & M_{nn} \end{pmatrix} \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix}$$

$$\vec{U} = -(M - I)^{-1}\vec{R}$$

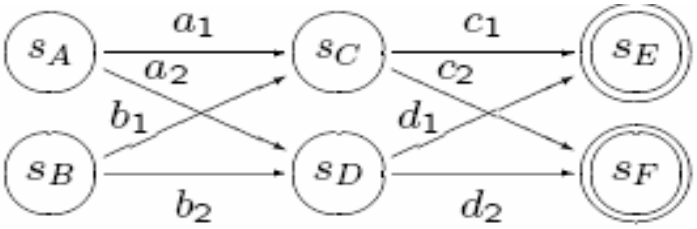
(Note: most $M_{ij} = 0$)

Q3: What if each state $\approx$ m features?

$\Rightarrow$ n = 2^m. . .

Q4: What if policy is NOT known?

Finding Best Policy



Let $U^*(s)$ $\equiv$ BEST possible, starting at s.

- Easy at end. . .

Time =	1	2	3
⋮			
s_D			
s_E			$U^*(s_E) = R(s_E) = +1$
s_F			$U^*(s_F) = R(s_F) = -1$

Q: What is **BEST** value of being in state s_D @ time=2?

A: Reward for being in s_D, PLUS value obtained by taking BEST action $\pi^*(s_D)$

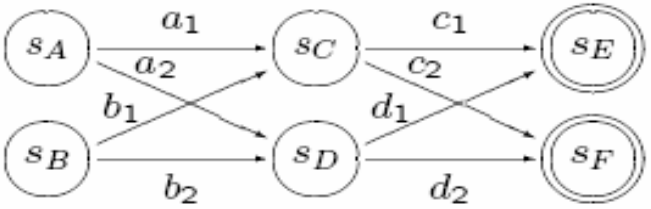
Best Action $\pi^*(s)$

- Q: What is best action $\pi^*(s)$?
- A: Consider single action $a \in \{ a_i \}$
- $P(s_j \mid s, a) \Rightarrow$ (distribution over) $\{ s_j \}$
  $\Rightarrow$ (distribution over) $\{ U^*(s_j) \}$
  $\Rightarrow$ each a_i has EXPECTED value
  $EU_s(a) = \sum_j P(s_j \mid s, a) \times U(s_j)$
- Take action $a^* = \pi^*(s) = \text{argmax}_a \{ EU_s(a) \}$

Eg: From S_D

- d_1 leads to S_E w/prob 1.0
  As $U(S_E) = -1$, $EU_{SD}(d_1) = 1.0 \times -1 = -1$
- d_2 leads to S_F w/prob 1.0
  As $U(S_F) = +1$, $EU_{SD}(d_2) = 1.0 \times +1 = +1$
- $\Rightarrow$ take $\pi^*(S_D) = d_2$ as $EU_{SD}(d_2) > EU_{SD}(d_1)$
- Similarly $\pi^*(S_C) = c_1$

Why Possible?



- $\pi^*(S_C) = c_1$, $\pi^*(S_D) = d_2$, $\pi^*(S_B) = b_1$, $\pi^*(S_A) = a_2$

  $U^*(S_E) = -1$, $U^*(S_F) = +1$, $U^*(S_C) = +1$, $U^*(S_D) = +1$, $U^*(S_A) = +1$, $U^*(S_B) = +1$

- Why possible?
  Know $U^*(s)$ for all "downstream" s's

Q: What if $\exists$ loops?

A: Same idea:

> Compute best action $\pi^*(s)$ for s
> assuming $U^*(s')$ for all "downstream" s' correct
>
> Then iterate...

Dynamic Programming

- Spse n-step process leads to terminals $\{S_n^{(k)}\}_k$

- Set $U^*(S_n^{(k)}) = R(S_n^{(k)})$
- Now work backwards:
  Compute $U^*(S_{n-1}^{(k')})$ for each $S_{n-1}^{(k')}$ one step from a $S_n^{(k)}$
  (ie, `Result`(a, S_{n-1}) includes S_n for some action a)

  $$U^*(S_{n-1}) = R(S_{n-1}) + \max_a \sum_j M^a_{n-1,j} U^*(S_n)$$

  $\pi^*(S_{n-1})$, policy at S_{n-1}, is that a in max

  Then compute $U(S_{n-2})$
  for each S_{n-2} one step from a S_{n-1}

  ...

  Until reaching S_0

Comments:
1. Exhaustive enumeration: $O(|A|^n)$
2. Dynamic Programming: $O(|A||S|n)$
   As need to consider each action, at each state, at each "depth"

$\Rightarrow$ Dynamic Programming works best when
- + $\exists$ SMALL number of possible STATES
- + know state exactly
- + known final depth n

Here: ... arbitrary depth