Decision Making under Uncertainty

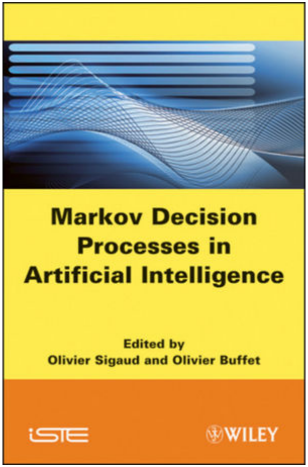
MDPs and POMDPs

Mykel J. Kochenderfer

27 January 2014

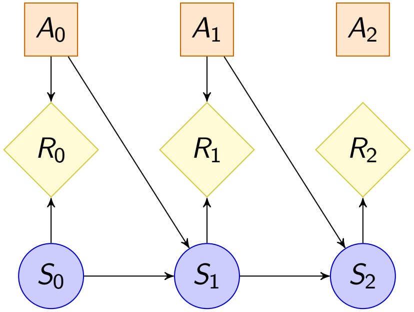
Recommended reference

Markov Decision Processes in Artificial Intelligence
edited by Sigaud and Buffet


Markov Decision Processes in Artificial Intelligence
Edited by
Olivier Sigaud and Olivier Buffet
iSTE
WILEY

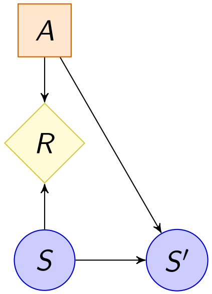
- Surveys a broad range of related topics
- Published in 2008, contains many recent references
- Contains psuedocode for algorithms
- More rigorous treatments can be found elsewhere

Stochastic problems



- Agent chooses action A_t at time t based on observing state S_t
- State evolves probabilistically based on current state and action taken by agent (Markov assumption)
- Objective is to maximize sum of rewards $R_1, \dots$

Markov decision process



- If transition and utility models are stationary (does not vary with time), then a dynamic Bayesian network can be used
- Problem is known as a Markov decision process (MDPs)
- Defined by transition model $T(s' \mid s, a)$
  and reward model $R(s, a)$ (often assumed deterministic)

Example Markov decision process

Aircraft collision avoidance

- Own aircraft must choose to stay level, climb, or descend
- At each step, -1 for collision, -0.01 for climb or descend
- State determined by altitude, closure rate, and vertical rates
- Intruder aircraft flies around randomly

Optimal behavior determined by reward and transition model

Models of optimal behavior

- In the finite-horizon model agent should optimize expected reward for the next H steps: $E(\sum_{t=0}^{H} r_t)$
- Continuously executing H-step optimal actions is known as receding horizon control
- In the infinite-horizon discounted model agent should optimize $E(\sum_{t=0}^{\infty} \gamma^t r_t)$
- Discount factor $0 \leq \gamma < 1$ can be thought of as an interest rate (reward now is worth more than reward in the future)
- Discounting keeps utility of an infinite sequence finite

Outline

Policies and utilities of states

- A policy π specifies what action to execute from every possible state
- Action to execute from state s according to π is denoted $\pi(s)$
- Expected utility of executing π when starting from s is denoted $U^{\pi}(s)$
- Optimal policy π^* is one that maximizes expected utility: $\pi^*(s) = \arg\max_{\pi} U^{\pi}(s)$

Iterative policy evaluation

- Incrementally compute expected utility after k steps of executing π
- $U_0^\pi(s) = 0$
- $U_1^\pi(s) = R(s, \pi(s))$
- ...
- $U_k^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U_{k-1}^\pi(s')$

This kind of iterative calculation is called dynamic programming

Policy evaluation

- For an infinite horizon,
  $U^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) U^\pi(s')$
- Can compute this arbitrarily well with enough iterations of iterative policy evaluation
- Alternative is to just solve system of N linear equations, where N is the number of states, requiring $O(n^3)$ time
- $U^\pi = R^\pi + \gamma T^\pi U^\pi$ (in matrix form)
- $U^\pi - \gamma T^\pi U^\pi = R^\pi$
- $(I - \gamma T^\pi) U^\pi = R^\pi$
- $U^\pi = (I - \gamma T^\pi)^{-1} R^\pi$
  (in Matlab, best to use matrix left division)

Policy iteration

- Policy iteration is one way to compute an optimal policy π^*
- The algorithm starts with any policy π_0 and iterates the following steps
  1. Policy evaluation: given π_i compute U^{π_i}
  2. Policy improvement: compute new policy from U^{π_i}
     $\pi_{i+1}(s) = \arg\max_a [R(s, a) + \gamma \sum_{s'} T(s' \mid s, a) U^{\pi_i}(s')]$
- Algorithm terminates when there is no more improvement
- Since every step leads to improvement and there are finitely many policies, algorithm terminates at optimal solution

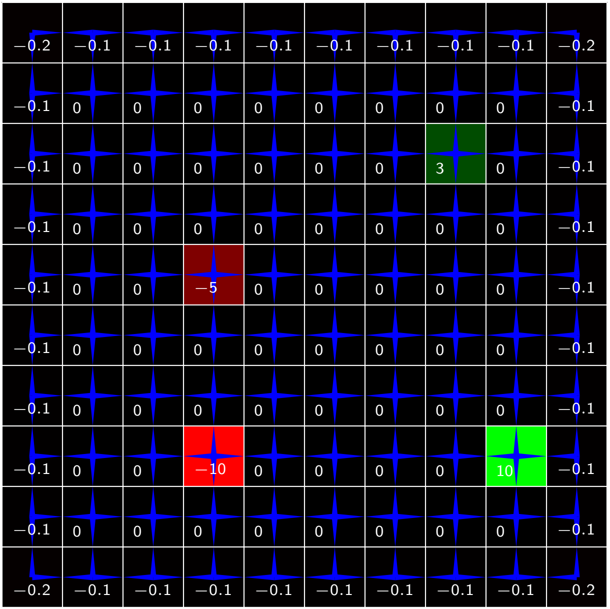
Value iteration

- An alternative algorithm is value iteration
- Bellman equation says value of optimal policy is given by $U^*(s) = \max_a[R(s,a) + \gamma \sum_{s'} T(s' \mid s,a)U^*(s')]$
- $U_0^*(s) = 0$ (initialization)
- $U_1^*(s) = \max_a[R(s,a) + \gamma \sum_{s'} T(s' \mid s,a)U_0^*(s')]$
- ...
- $U_k^*(s) = \max_a[R(s,a) + \gamma \sum_{s'} T(s' \mid s,a)U_{k-1}^*(s')]$
- $U_k^* \to U^*$ as $k \to \infty$
- $\pi(s) = \arg\max_a[R(s,a) + \gamma \sum_{s'} T(s' \mid s,a)U(s')]$

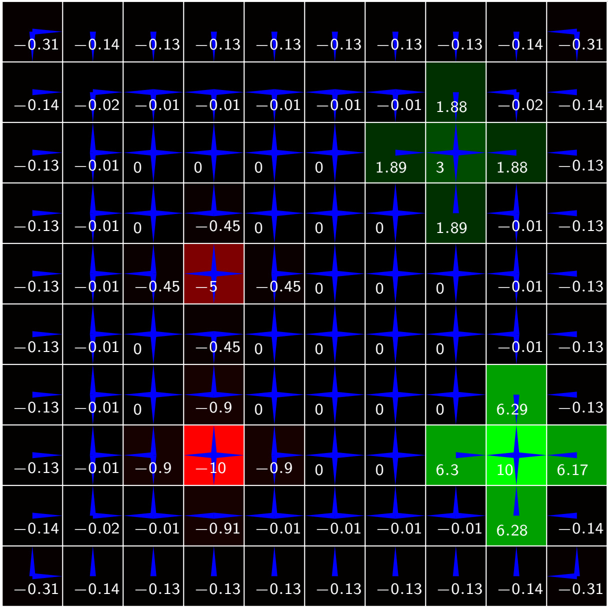
Example grid world

- States: cells in 10×10 grid
- Actions: up, down, left, and right
- Transition model: 0.7 chance of moving in intended direction, uniform in other directions
- Reward: two states have cost (and are not terminal), two have reward (and are terminal), -1 for wall crash

Grid world: Iteration 1 ($\gamma = 0.9$)

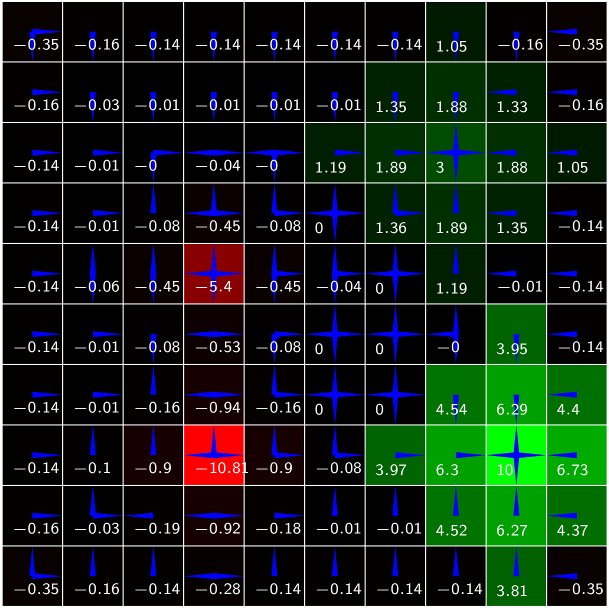


Grid world: Iteration 2 ($\gamma = 0.9$)



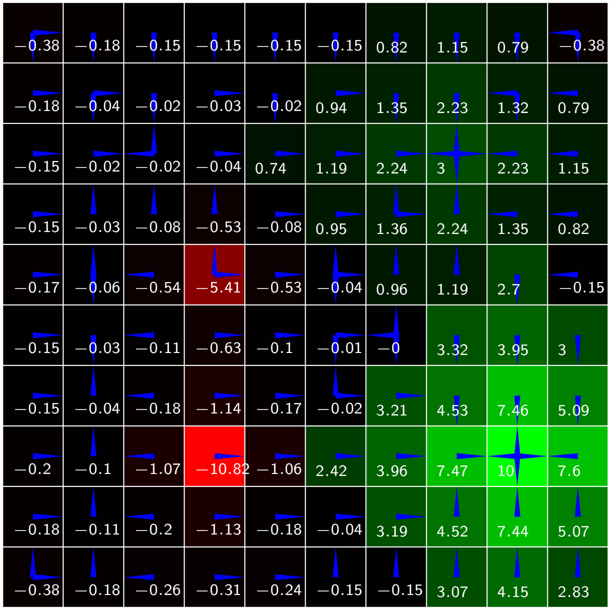
−0.31	−0.14	−0.13	−0.13	−0.13	−0.13	−0.13	−0.13	−0.14	−0.31
−0.14	−0.02	−0.01	−0.01	−0.01	−0.01	−0.01	1.88	−0.02	−0.14
−0.13	−0.01	0	0	0	0	1.89	3	1.88	−0.13
−0.13	−0.01	0	−0.45	0	0	0	1.89	−0.01	−0.13
−0.13	−0.01	−0.45	−5	−0.45	0	0	0	−0.01	−0.13
−0.13	−0.01	0	−0.45	0	0	0	0	−0.01	−0.13
−0.13	−0.01	0	−0.9	0	0	0	0	6.29	−0.13
−0.13	−0.01	−0.9	−10	−0.9	0	0	6.3	10	6.17
−0.14	−0.02	−0.01	−0.91	−0.01	−0.01	−0.01	−0.01	6.28	−0.14
−0.31	−0.14	−0.13	−0.13	−0.13	−0.13	−0.13	−0.13	−0.14	−0.31

Grid world: Iteration 3 ($\gamma = 0.9$)

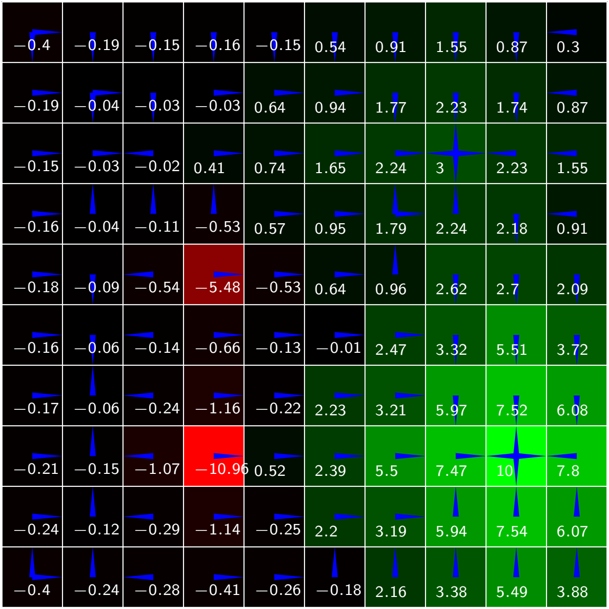


−0.35	−0.16	−0.14	−0.14	−0.14	−0.14	−0.14	1.05	−0.16	−0.35
−0.16	−0.03	−0.01	−0.01	−0.01	−0.01	1.35	1.88	1.33	−0.16
−0.14	−0.01	−0	−0.04	−0	1.19	1.89	3	1.88	1.05
−0.14	−0.01	−0.08	−0.45	−0.08	0	1.36	1.89	1.35	−0.14
−0.14	−0.06	−0.45	−5.4	−0.45	−0.04	0	1.19	−0.01	−0.14
−0.14	−0.01	−0.08	−0.53	−0.08	0	0	−0	3.95	−0.14
−0.14	−0.01	−0.16	−0.94	−0.16	0	0	4.54	6.29	4.4
−0.14	−0.1	−0.9	−10.81	−0.9	−0.08	3.97	6.3	10	6.73
−0.16	−0.03	−0.19	−0.92	−0.18	−0.01	−0.01	4.52	6.27	4.37
−0.35	−0.16	−0.14	−0.28	−0.14	−0.14	−0.14	−0.14	3.81	−0.35

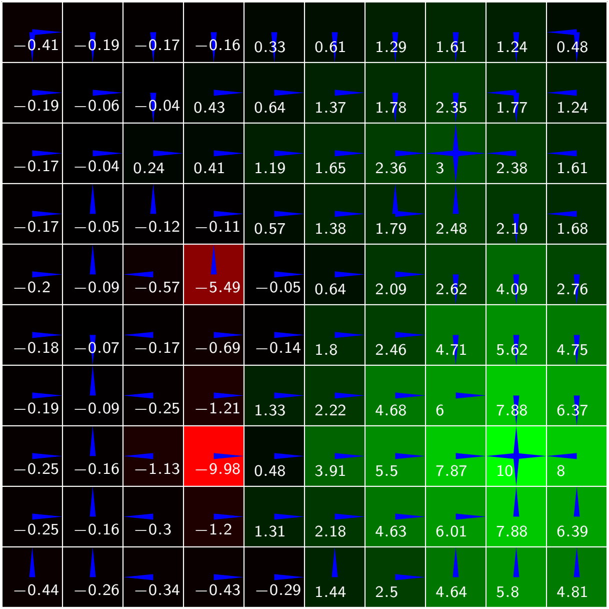
Grid world: Iteration 4 ($\gamma = 0.9$)



Grid world: Iteration 5 ($\gamma = 0.9$)

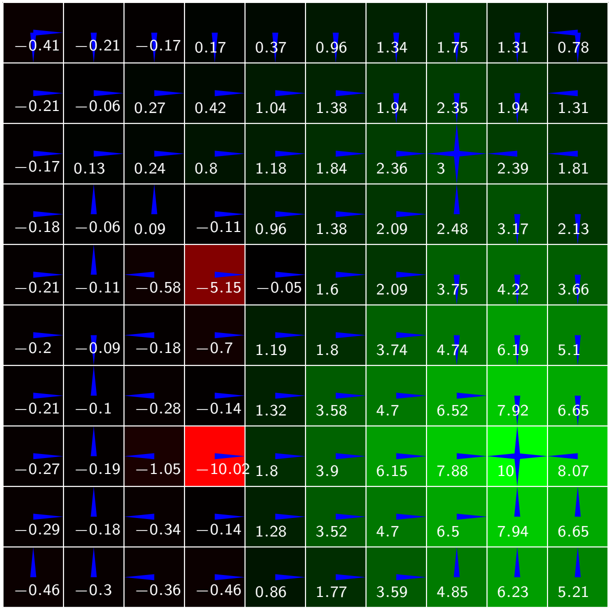


Grid world: Iteration 6 ($\gamma = 0.9$)



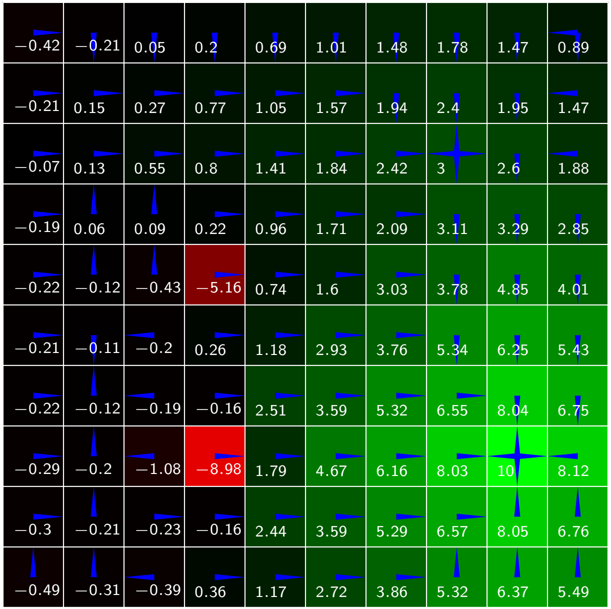
−0.41	−0.19	−0.17	−0.16	0.33	0.61	1.29	1.61	1.24	0.48
−0.19	−0.06	−0.04	0.43	0.64	1.37	1.78	2.35	1.77	1.24
−0.17	−0.04	0.24	0.41	1.19	1.65	2.36	3	2.38	1.61
−0.17	−0.05	−0.12	−0.11	0.57	1.38	1.79	2.48	2.19	1.68
−0.2	−0.09	−0.57	−5.49	−0.05	0.64	2.09	2.62	4.09	2.76
−0.18	−0.07	−0.17	−0.69	−0.14	1.8	2.46	4.71	5.62	4.75
−0.19	−0.09	−0.25	−1.21	1.33	2.22	4.68	6	7.88	6.37
−0.25	−0.16	−1.13	−9.98	0.48	3.91	5.5	7.87	10	8
−0.25	−0.16	−0.3	−1.2	1.31	2.18	4.63	6.01	7.88	6.39
−0.44	−0.26	−0.34	−0.43	−0.29	1.44	2.5	4.64	5.8	4.81

Grid world: Iteration 7 ($\gamma = 0.9$)



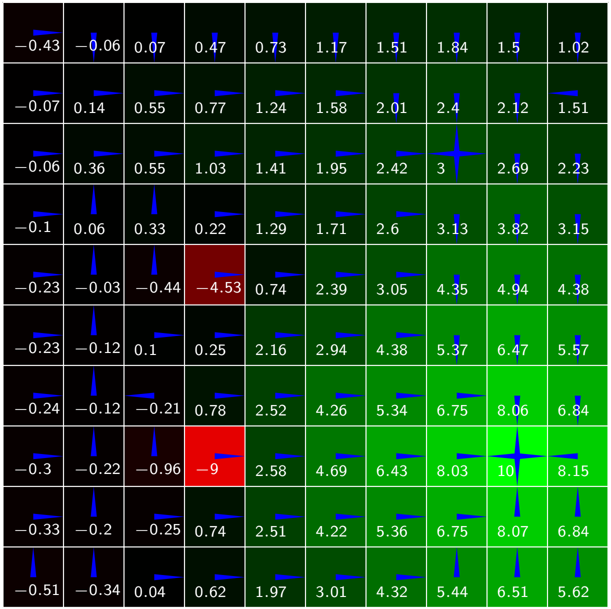
−0.41	−0.21	−0.17	0.17	0.37	0.96	1.34	1.75	1.31	0.78
−0.21	−0.06	0.27	0.42	1.04	1.38	1.94	2.35	1.94	1.31
−0.17	0.13	0.24	0.8	1.18	1.84	2.36	3	2.39	1.81
−0.18	−0.06	0.09	−0.11	0.96	1.38	2.09	2.48	3.17	2.13
−0.21	−0.11	−0.58	−5.15	−0.05	1.6	2.09	3.75	4.22	3.66
−0.2	−0.09	−0.18	−0.7	1.19	1.8	3.74	4.74	6.19	5.1
−0.21	−0.1	−0.28	−0.14	1.32	3.58	4.7	6.52	7.92	6.65
−0.27	−0.19	−1.05	−10.02	1.8	3.9	6.15	7.88	10	8.07
−0.29	−0.18	−0.34	−0.14	1.28	3.52	4.7	6.5	7.94	6.65
−0.46	−0.3	−0.36	−0.46	0.86	1.77	3.59	4.85	6.23	5.21

Grid world: Iteration 8 ($\gamma = 0.9$)



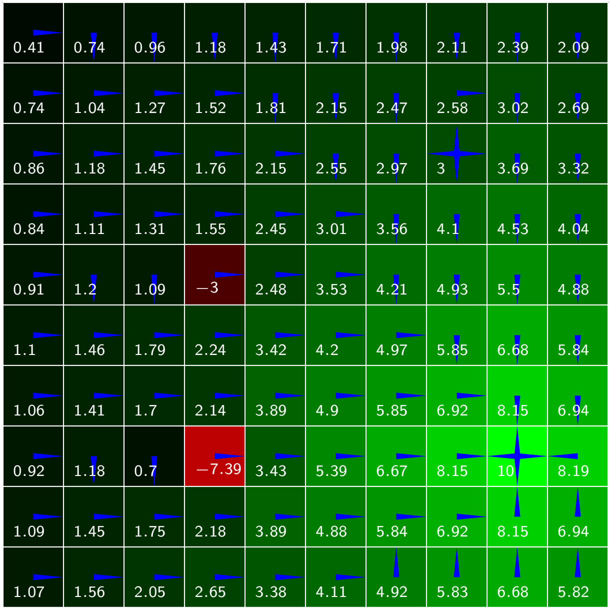
-0.42	-0.21	0.05	0.2	0.69	1.01	1.48	1.78	1.47	0.89
-0.21	0.15	0.27	0.77	1.05	1.57	1.94	2.4	1.95	1.47
-0.07	0.13	0.55	0.8	1.41	1.84	2.42	3	2.6	1.88
-0.19	0.06	0.09	0.22	0.96	1.71	2.09	3.11	3.29	2.85
-0.22	-0.12	-0.43	-5.16	0.74	1.6	3.03	3.78	4.85	4.01
-0.21	-0.11	-0.2	0.26	1.18	2.93	3.76	5.34	6.25	5.43
-0.22	-0.12	-0.19	-0.16	2.51	3.59	5.32	6.55	8.04	6.75
-0.29	-0.2	-1.08	-8.98	1.79	4.67	6.16	8.03	10	8.12
-0.3	-0.21	-0.23	-0.16	2.44	3.59	5.29	6.57	8.05	6.76
-0.49	-0.31	-0.39	0.36	1.17	2.72	3.86	5.32	6.37	5.49

Grid world: Iteration 9 ($\gamma = 0.9$)



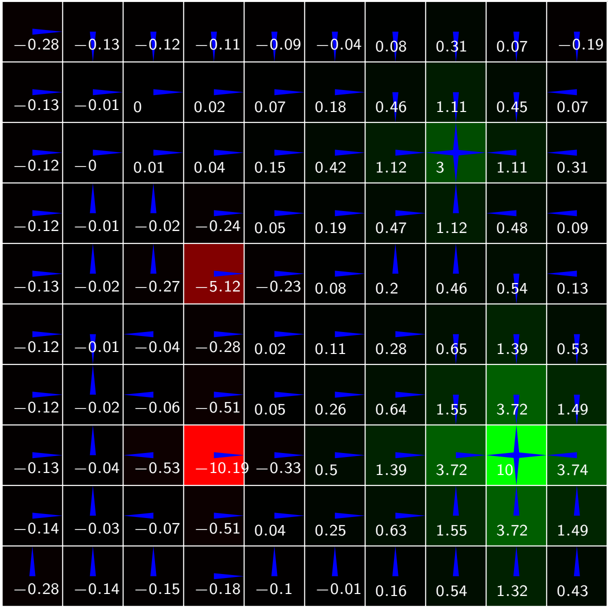
−0.43	−0.06	0.07	0.47	0.73	1.17	1.51	1.84	1.5	1.02
−0.07	0.14	0.55	0.77	1.24	1.58	2.01	2.4	2.12	1.51
−0.06	0.36	0.55	1.03	1.41	1.95	2.42	3	2.69	2.23
−0.1	0.06	0.33	0.22	1.29	1.71	2.6	3.13	3.82	3.15
−0.23	−0.03	−0.44	−4.53	0.74	2.39	3.05	4.35	4.94	4.38
−0.23	−0.12	0.1	0.25	2.16	2.94	4.38	5.37	6.47	5.57
−0.24	−0.12	−0.21	0.78	2.52	4.26	5.34	6.75	8.06	6.84
−0.3	−0.22	−0.96	−9	2.58	4.69	6.43	8.03	10	8.15
−0.33	−0.2	−0.25	0.74	2.51	4.22	5.36	6.75	8.07	6.84
−0.51	−0.34	0.04	0.62	1.97	3.01	4.32	5.44	6.51	5.62

Grid world: Converged ($\gamma = 0.9$)



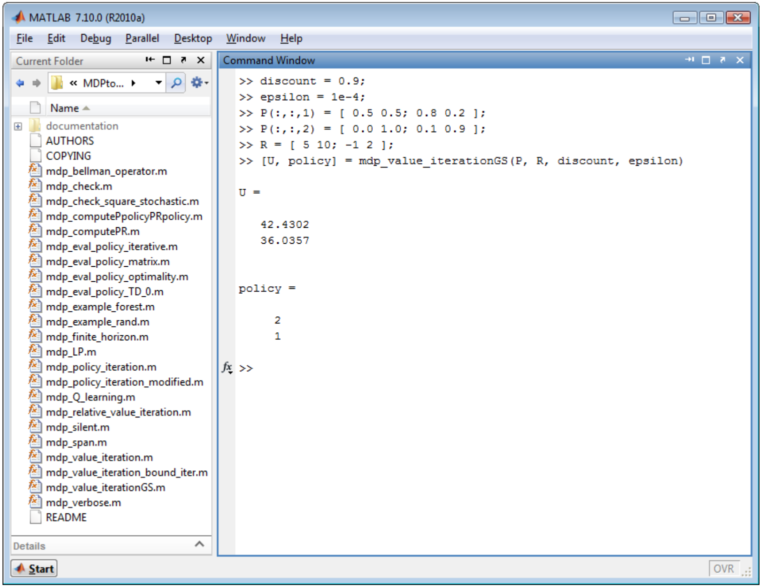
0.41	0.74	0.96	1.18	1.43	1.71	1.98	2.11	2.39	2.09
0.74	1.04	1.27	1.52	1.81	2.15	2.47	2.58	3.02	2.69
0.86	1.18	1.45	1.76	2.15	2.55	2.97	3	3.69	3.32
0.84	1.11	1.31	1.55	2.45	3.01	3.56	4.1	4.53	4.04
0.91	1.2	1.09	-3	2.48	3.53	4.21	4.93	5.5	4.88
1.1	1.46	1.79	2.24	3.42	4.2	4.97	5.85	6.68	5.84
1.06	1.41	1.7	2.14	3.89	4.9	5.85	6.92	8.15	6.94
0.92	1.18	0.7	-7.39	3.43	5.39	6.67	8.15	10	8.19
1.09	1.45	1.75	2.18	3.89	4.88	5.84	6.92	8.15	6.94
1.07	1.56	2.05	2.65	3.38	4.11	4.92	5.83	6.68	5.82

Grid world: Converged ($\gamma = 0.5$)



−0.28	−0.13	−0.12	−0.11	−0.09	−0.04	0.08	0.31	0.07	−0.19
−0.13	−0.01	0	0.02	0.07	0.18	0.46	1.11	0.45	0.07
−0.12	−0	0.01	0.04	0.15	0.42	1.12	3	1.11	0.31
−0.12	−0.01	−0.02	−0.24	0.05	0.19	0.47	1.12	0.48	0.09
−0.13	−0.02	−0.27	−5.12	−0.23	0.08	0.2	0.46	0.54	0.13
−0.12	−0.01	−0.04	−0.28	0.02	0.11	0.28	0.65	1.39	0.53
−0.12	−0.02	−0.06	−0.51	0.05	0.26	0.64	1.55	3.72	1.49
−0.13	−0.04	−0.53	−10.19	−0.33	0.5	1.39	3.72	10	3.74
−0.14	−0.03	−0.07	−0.51	0.04	0.25	0.63	1.55	3.72	1.49
−0.28	−0.14	−0.15	−0.18	−0.1	−0.01	0.16	0.54	1.32	0.43

Markov Decision Process (MDP) Toolbox for Matlab



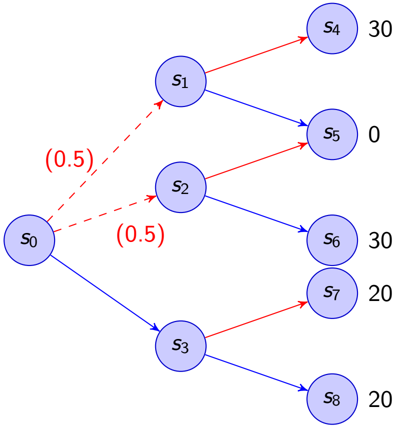
```
documentation
AUTHORS
COPYING
mdp_bellman_operator.m
mdp_check.m
mdp_check_square_stochastic.m
mdp_computePpolicyPRpolicy.m
mdp_computePR.m
mdp_eval_policy_iterative.m
mdp_eval_policy_matrix.m
mdp_eval_policy_optimality.m
mdp_eval_policy_TD_0.m
mdp_example_forest.m
mdp_example_rand.m
mdp_finite_horizon.m
mdp_LP.m
mdp_policy_iteration.m
mdp_policy_iteration_modified.m
mdp_Q_learning.m
mdp_relative_value_iteration.m
mdp_silent.m
mdp_span.m
mdp_value_iteration.m
mdp_value_iteration_bound_iter.m
mdp_value_iterationGS.m
mdp_verbose.m
README
```

```
>> discount = 0.9;
>> epsilon = 1e-4;
>> P(:,:,1) = [ 0.5 0.5; 0.8 0.2 ];
>> P(:,:,2) = [ 0.0 1.0; 0.1 0.9 ];
>> R = [ 5 10; -1 2 ];
>> [U, policy] = mdp_value_iterationGS(P, R, discount, epsilon)

U =

   42.4302
   36.0357


policy =

     2
     1

>>
```

http://www.mathworks.com/matlabcentral/fileexchange/25786-markov-decision-processes-mdp-toolbox

Closed loop and open loop planning

- Closed loop: accounts for future state information (MDP)
- Open loop: does not account for future state information (path planning)



- Open loop plans do not always result in optimal behavior
- $U(r, r) = 15$
- $U(r, b) = 15$
- $U(b, r) = 20$
- $U(b, b) = 20$
- MDP solution can increase utility to 30

Outline

Factored Markov decision processes

- Number of discrete states grows exponentially with the number of state variables
- Factored Markov decision processes (FMDPs) compactly represent transition and reward functions using dynamic decision networks
- Conditional probability distributions and utility functions can be represented as decision trees

Coffee robot problem

- Robot must go buy a cup of coffee for its owner located at the office
- When it's raining, robot must get an umbrella to stay dry
- Actions: go to other location, buy coffee, deliver coffee, and get umbrella

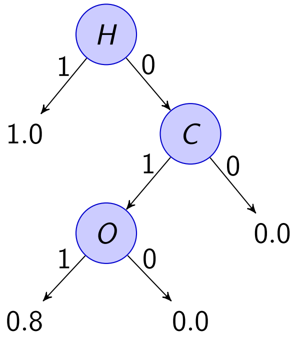
Transition model for deliver coffee action

$$\begin{tikzcd}
W \arrow[r] & W & \text{wet?} \\
U \arrow[r] & U & \text{umbrella?} \\
R \arrow[r] & R & \text{raining?} \\
O \arrow[r] \arrow[dr] \arrow[ddr] & O & \text{office?} \\
H \arrow[r] & H & \text{owner have coffee?} \\
C \arrow[r] \arrow[ur] & C & \text{robot have coffee?}
\end{tikzcd}$$

Conditional distribution as decision tree

Representation of $P(H' = 1 \mid H, C, O, \text{deliver coffee})$

H	C	O	$P(H')$
1	1	1	1.0
1	1	0	1.0
1	0	1	1.0
1	0	0	1.0
0	1	1	0.8
0	1	0	0.0
0	0	1	0.0
0	0	0	0.0



Both the table and decision tree represent the same distribution, but the decision tree is more compact

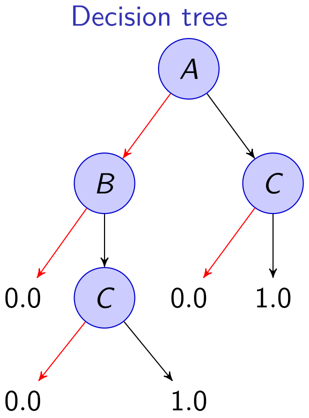
Structured dynamic programming

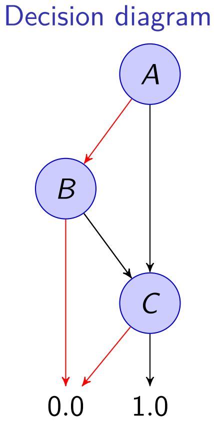
- $R(s, a)$ and $U(s)$ can also be represented using a decision tree
- Structured value iteration and structured policy iteration performs updates on leaves of the decision trees instead of all the states
- Structured dynamic programming algorithms improve efficiency by aggregating states
- Algorithms can also leverage additive decomposition of reward and value function

DMU 4.3.2, C. Boutilier, R. Dearden, and M. Goldszmidt, "Stochastic dynamic programming with factored representations," *Artificial Intelligence*, vol. 121, no. 1–2, pp. 49–107, Aug. 2000. DOI: 10.1016/S0004-3702(00)00033-3

Stochastic Planning Using Decision Diagrams (SPUDD)

Additional efficiency can be gained using decision diagrams





Software: `http://www.computing.dundee.ac.uk/staff/jessehoey/spudd/index.html`

DMU 4.3.2, J. Hoey, R. St-Aubin, A. Hu, *et al.*, "SPUDD: Stochastic planning using decision diagrams," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, K. B. Laskey and H. Prade, Eds., Morgan Kaufmann, 1999, pp. 279–288

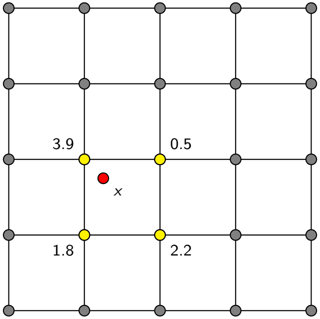
Outline

Approximate dynamic programming

- Discussion up to this point assumed (small) discrete state and action spaces
- Approximate dynamic programming is concerned with finding approximately optimal policies for problems with large or continuous spaces
- ADP is an active area of research and shares ideas with reinforcement learning
- Methods include parametric approximation, local approximation, and state aggregation (abstraction) methods

Grid-based parametric approximation

- If state space X is continuous, how to represent utility U?
- One way is to use a grid to get a discrete set of points $S \subset X$
- Let V be a utility function over S
- Determine $U(x)$ by interpolating V
- Bellman update:
  $V_k(s) \leftarrow \max_a[R(s,a) + \gamma \sum_{x'} P(x' \mid s,a) U_{k-1}(x')]$



DMU 4.5.1, G. J. Gordon, "Approximate solutions to Markov decision processes," PhD thesis, Computer Science Department, Carnegie Mellon University, 1999

Linear regression parametric approximation

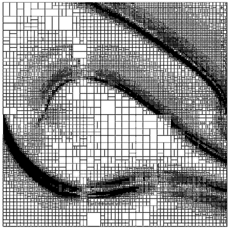
- Instead of discretizing the state space (exponential in state variables) define a set of basis functions $\phi_1, \ldots, \phi_n$ over X
- Each basis function maps states in X to real numbers
- Think of basis functions as a state-dependent features
- Represent U as a linear combination of basis functions: $U(x) = \theta_1 \phi_1(x) + \ldots + \theta_n \phi_n(x)$
- Given fixed basis functions, determine $\theta_1, \ldots, \theta_n$ that best represents the optimal value function

Linear regression parametric approximation

- Let S be a finite set of samples from X (doesn't have to be from a grid)
- Initialize $\theta_1, \ldots, \theta_n$ to 0, making $U_0(x) = 0$
- Bellman update:
  $V_k(s) \leftarrow \max_a [R(s,a) + \gamma \sum_{x'} P(x' \mid s,a) U_{k-1}(x')]$
- Use regression (e.g., least squares) to find U_k that approximates V_k at sample points in S
- Approach can be generalized to other parametric approximation methods (e.g., neural networks)

Variable resolution dynamic programming

- Carve up X into a discrete state space S
- Instead of solving for the optimal policy over X, solve for policy over S
- Some parts of the state space require finer resolution than other parts when finding a good policy
- One approach is to adaptively partition the state space according to heuristics



R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2–3, pp. 291–323, 2002. DOI: doi:10.1023/A:1017992615625

Linear representations

Exact solutions for continuous state and action spaces are available under certain assumptions:

- Dynamics are linear Gaussian:
  $T(\mathbf{z} \mid \mathbf{s}, \mathbf{a}) = \mathcal{N}(\mathbf{z} \mid \mathbf{T}_s\mathbf{s} + \mathbf{T}_a\mathbf{a}, \Sigma)$
- Rewards are quadratic:
  $R(\mathbf{s}, \mathbf{a}) = \mathbf{s}^\top \mathbf{R}_s \mathbf{s} + \mathbf{a}^\top \mathbf{R}_a \mathbf{a}$
  assuming $\mathbf{R}_s = \mathbf{R}_s^\top \leq 0$ and $\mathbf{R}_a = \mathbf{R}_a^\top < 0$

Bellman equation (undiscounted) is:
$U_n(\mathbf{s}) = \max_{\mathbf{a}} \left(R(\mathbf{s}, \mathbf{a}) + \int T(\mathbf{z} \mid \mathbf{s}, \mathbf{a}) U_{n-1}(\mathbf{z}) d\mathbf{z} \right)$

Linear representations

- It is possible to show that
  $U_n(\mathbf{s}) = \mathbf{s}^\top \mathbf{V}_n \mathbf{s} + q_n$
  $\mathbf{V}_n = \mathbf{T}_s^\top \mathbf{V}_{n-1} \mathbf{T}_s - \mathbf{T}_s^\top \mathbf{V}_{n-1} \mathbf{T}_a (\mathbf{T}_a^\top \mathbf{T}_a + \mathbf{R}_a)^{-1} \mathbf{T}_s^\top \mathbf{V}_{n-1} \mathbf{T}_s + \mathbf{R}_s$
  $q_n = q_{n-1} + \mathrm{Tr}(\Sigma \mathbf{V}_{n-1})$ assuming $\mathbf{R}_s = \mathbf{R}_s^\top \leq 0$ and $\mathbf{R}_a = \mathbf{R}_a^\top < 0$
- Optimal policy:
  $\pi_n(\mathbf{s}) = -(\mathbf{T}_a^\top \mathbf{V}_{n-1} \mathbf{T}_a + \mathbf{R}_a)^{-1} \mathbf{T}_a \mathbf{V}_{n-1} \mathbf{T}_s \mathbf{s}$
- Note that optimal policy does not depend upon noise!
- For some problems, the dynamics are approximated with a linear Gaussian system and then solved exactly

Direct policy search

- Alternative to planning over entire state space is to search the space of policies directly
- Consider stochastic policy parameterized by $\boldsymbol{\lambda}$
- Probability policy selects a in state s is given by $\pi_{\boldsymbol{\lambda}}(a \mid s)$
- Use Monte Carlo simulations to estimate expected discounted return from a distribution of initial states
- Find best $\boldsymbol{\lambda}$ using local search, cross entropy method, or evolutionary algorithms

Outline

Online methods

- Online methods compute optimal action from current state
- Expand tree up to some horizon
- States reachable from the current state is typically small compared to full state space
- Heuristics and branch-and-bound techniques allow search space to be pruned
- Monte Carlo methods provide approximate solutions

Forward search

Provides optimal action from current state s up to depth d

Recall $U(s) = \max_{a \in A(s)}[R(s,a) + \gamma \sum_{s' \in S(s,a)} T(s' \mid s,a)U(s')]$

```
1:  function $\textsc{SelectAction}(s, d)$
2:      if $d = 0$
3:          return $(\textsc{nil}, 0)$
4:      $(a^*, v^*) \leftarrow (\textsc{nil}, -\infty)$
5:      for $a \in A(s)$
6:          $q \leftarrow R(s, a)$
7:          for $s' \in S(s, a)$
8:              $(a', v') \leftarrow \textsc{SelectAction}(s', d - 1)$
9:              $q \leftarrow q + \gamma T(s' \mid s, a)v'$
10:         if $q > v^*$
11:             $(a^*, v^*) \leftarrow (a, q)$
12:     return $(a^*, v^*)$
```

Time complexity is $O((|S| \times |A|)^d)$

Branch and bound search

Requires a lower bound $\underline{U}(s)$ and upper bound $\overline{U}(s, a)$

```
1:  function $\textsc{SelectAction}(s, d)$
2:      if $d = 0$
3:          return $(\textsc{nil}, \underline{U}(s))$
4:      $(a^*, v^*) \leftarrow (\textsc{nil}, -\infty)$
5:      for $a \in A(s)$
6:          if $\overline{U}(s, a) < v^*$
7:              return $(a^*, v^*)$
8:          $q \leftarrow R(s, a)$
9:          for $s' \in S(s, a)$
10:             $(a', v') \leftarrow \textsc{SelectAction}(s', d - 1)$
11:             $q \leftarrow q + \gamma T(s' \mid s, a) v'$
12:         if $q > v^*$
13:             $(a^*, v^*) \leftarrow (a, q)$
14:     return $(a^*, v^*)$
```

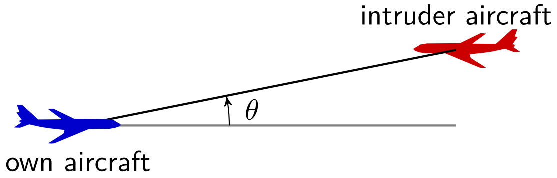
Sparse sampling

Requires a generative model $(s', r) \sim G(s, a)$

```
1:  function SelectAction($s, d$)
2:      if $d = 0$
3:          return $(\text{NIL}, 0)$
4:      $(a^*, v^*) \leftarrow (\text{NIL}, -\infty)$
5:      for $a \in A(s)$
6:          $q \leftarrow 0$
7:          for $i \leftarrow 1$ to $n$
8:              $(s', r) \sim G(s, a)$
9:              $(a', v') \leftarrow \text{SelectAction}(s', d - 1)$
10:             $q \leftarrow q + (r + \gamma v')/n$
11:         if $q > v^*$
12:             $(a^*, v^*) \leftarrow (a, q)$
13:     return $(a^*, v^*)$
```

Example POMDP

Aircraft collision avoidance using an electro-optical sensor

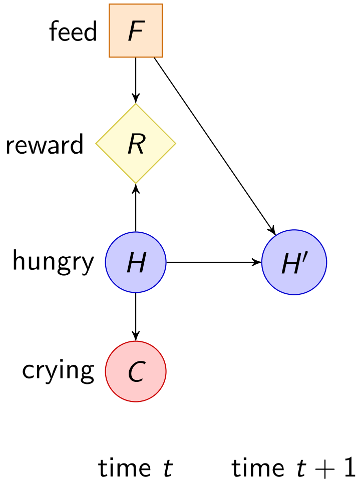


- Own aircraft can accelerate up or down
- Large penalty for collision, small penalty for maneuvering
- Dynamics are a function of current position and velocity
- Electro-optical sensor only measures angle
- Limited field of view

Optimization resulted in “passive ranging” behavior.

S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, “Collision Avoidance for Unmanned Aircraft using Markov Decision Processes,” in AIAA Guidance, Navigation, and Control Conference, Toronto, Canada, 2010.

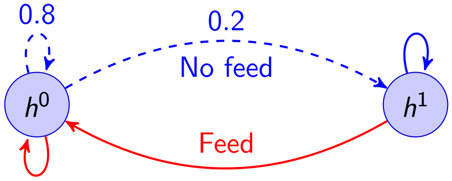
Crying baby problem



- Need to decide whether to feed baby given whether baby is crying
- Crying is a noisy indication that the baby is hungry
- Costs 5 to feed and 10 for hungry
- Infinite horizon with discount γ

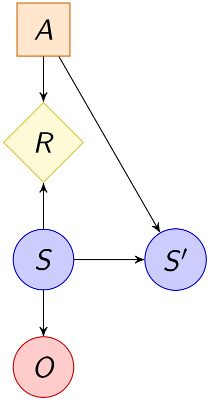
Crying baby problem

Transition model



- $P(c^1 \mid h^0) = 0.1$ (cry when not hungry)
- $P(c^1 \mid h^1) = 0.8$ (cry when hungry)

POMDPs



- POMDP = MDP + sensor model
- Sensor model: $O(o \mid s)$ or sometimes $O(o \mid s, a)$
- Decisions can only be based on history of observations $o_1, o_2, \ldots, o_t$
- Instead of keeping track of arbitrarily long histories, we keep track of the belief state
- A belief state is a distribution over states; in belief state b, probability $b(s)$ is assigned to being in s

Policies in POMDPs are mappings from belief states to actions

Computing belief states

- Begin with some initial belief state b prior to any observations
- Compute new belief state b' based on current belief state b, action a, and observation o
- $b'(s') = P(s' \mid o, a, b)$
- $b'(s') \propto P(o \mid s', a, b)P(s' \mid a, b)$
- $b'(s') \propto O(o \mid s', a)P(s' \mid a, b)$
- $b'(s') \propto O(o \mid s', a) \sum_s P(s' \mid a, b, s)P(s \mid a, b)$
- $b'(s') \propto O(o \mid s', a) \sum_s T(s' \mid s, a)b(s)$
- Kalman filter: exact update of the belief state for linear dynamical systems
- Particle filter: approximate update for general systems
- For now, we're only considering discrete state problems

Crying baby example

- $b = (h^0, h^1) = (0.5, 0.5)$
- No feed, cry
- $b = (0.0928, 0.9072)$
- Feed, no cry
- $b = (1, 0)$
- No feed, no cry
- $b = (0.9759, 0.0241)$
- No feed, no cry
- $b = (0.9701, 0.0299)$
- No feed, cry
- $b = (0.4624, 0.5376)$

POMDP execution

1. Initialize belief state b
2. Execute $a = \pi(b)$
3. Observe o
4. Update b based on b, a, and o
5. Go to 2

POMDP as belief MDP

- POMDP is really an MDP where the states are belief states
- $\mathcal{B}$, the set of belief states, comprise the state space
    - If there are n discrete states, then $\mathcal{B} \subset \mathbb{R}^n$
- $\mathcal{A}$, the set of actions, remains the same
- $\tau(b' \mid b, a)$ is the state transition function
    - See DMU equation 6.5
- $\rho(b, a)$, the immediate reward, is equal to $\sum_s b(s)R(s, a)$

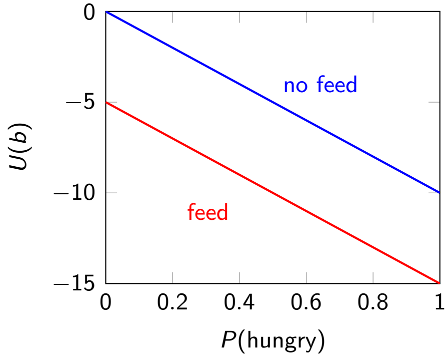
Discrete MDP methods cannot be used directly for POMDPs

DMU 6.1.4, AIMA 17.4.1, L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1–2, pp. 99–134, May 1998. DOI: doi:10.1016/S0004-3702(98)00023-X

Outline

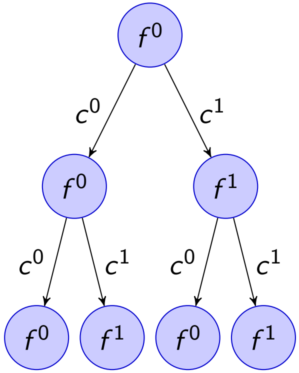
Alpha vectors

- For now, assume a 1-step horizon
- $U(s) = \max_a R(s, a)$
- $U(b) = \max_a \sum_s b(s) R(s, a)$
- Let α_a be $R(\cdot, a)$ represented as a vector
- $U(b) = \max_a b \cdot \alpha_a$
- The alpha vectors define hyperplanes in belief space
- One-step utility is piecewise linear and convex
- What about H-step utility?



Conditional plans

Example 3-step conditional plan

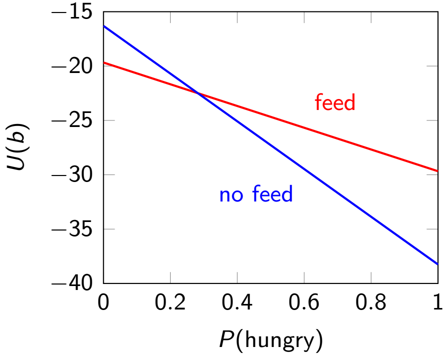


- A conditional plan specifies what to do from the initial belief state after each possible observation up to some horizon
- Let α_p represent the expected utility of following plan p
- $U(b) = \max_p b \cdot \alpha_p$
- H-step utility is also piecewise linear and convex
- Execute action at root node of plan that maximizes $b \cdot \alpha_p$

POMDP value iteration

- Value iteration for POMDPs involves generating conditional plans and associated alpha vectors up to some depth
- Number of possible conditional plans is doubly exponential
- For the crying baby problem, there are 2^{255} conditional 8-step plans
- During value iteration, toss out dominated plans

Optimal crying baby policy



Exact solution for general POMDP is PSPACE-hard

Outline

Offline methods

- Offline POMDP methods involve doing all or most of the computing prior to execution
- Practical method generally find only approximate solutions
- Some methods output policy in terms of alpha-vectors, others as finite-state controllers
- Some methods involve leveraging a factored representation
- Very active area of research

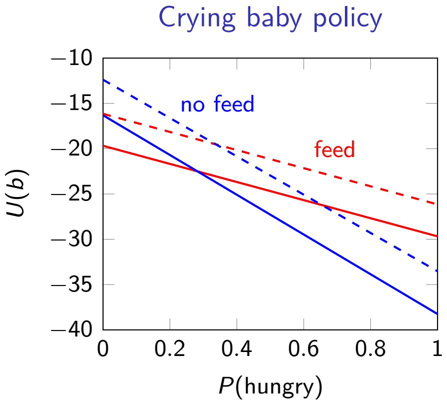
QMDP value approximation

- QMDP approximation method involves solving for the MDP assuming full observability
- Compute $Q(s, a)$ for all states and actions
- Let α_a be the vector representation of the Q function for action a
- QMDP utility function: $U_{\text{QMDP}}(b) = \max_a b \cdot \alpha_a$
- QMDP policy: $\pi_{\text{QMDP}}(b) = \arg\max_a b \cdot \alpha_a$

QMDP assumes all state uncertainty disappears at the next step

QMDP value approximation

- Upper bound on utility: $U_{\text{QMDP}}(b) \geq U^*(b)$
- Very small change in policy for crying baby problem
- QMDP tends to have difficulty with problems with information gathering actions (e.g., look over right shoulder when changing lanes)
- Performs extremely well in many real problems (e.g., airborne collision avoidance)



Dashed lines are for QMDP

Fast informed bound (FIB) value approximation

- FIB computes single alpha vector for each action (just like QMDP)
- Tighter bounds are provided by taking into account observation model
- QMDP:
  $\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) \max_{a'} \alpha_{a'}^{(k)}(s')$
- FIB:
  $\alpha_a^{(k+1)}(s) = R(s,a) + \gamma \sum_{o} \max_{a'} \sum_{s'} O(o \mid s',a) T(s' \mid s,a) \alpha_{a'}^{(k)}(s')$
- FIB complexity is $O(|A|^2|S|^2|O|)$
- FIB is also an upper bound on optimal value function

Point-based value iteration methods

- Point-based value iteration involves backing up alpha vectors $\Gamma = \boldsymbol{\alpha}_{1:n}$ associated with belief points $\mathbf{b}_{1:n}$
- Choose points $\mathbf{b}_{1:n}$ and initialize alpha vectors to lower bound
- Update alpha vectors in Γ at those points such that $U^{\Gamma}(\mathbf{b}) \leq U^{*}(\mathbf{b})$

```
1: function BackupBelief($\Gamma, \mathbf{b}$)
2:     for $a \in A$
3:         for $o \in O$
4:             $\mathbf{b}' \leftarrow$ UpdateBelief($\mathbf{b}, a, o$)
5:             $\boldsymbol{\alpha}_{a,o} \leftarrow \arg\max_{\boldsymbol{\alpha} \in \Gamma} \boldsymbol{\alpha}^{\top} \mathbf{b}'$
6:         $\alpha_a(s) \leftarrow R(s, a) + \gamma \sum_{s', o} O(o \mid s', a) T(s' \mid s, a) \alpha_{a,o}(s')$
7:     $\boldsymbol{\alpha} \leftarrow \arg\max_{\boldsymbol{\alpha}_a} \boldsymbol{\alpha}_a^{\top} \mathbf{b}$
8:     return $\boldsymbol{\alpha}$
```

Point-based value iteration methods

- Some algorithms use a fixed grid of belief states
- Better to focus effort on reachable belief states under random policies
- Even better to focus effort on belief states reachable under the optimal policy
- Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) algorithm is one of the best approximate solvers for POMDPs

DMU 6.4.3, MDPAI 7.5, 7.6, AIMA 17.7, H. Kurniawati, D. Hsu, and W. Lee, "SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008

APPL: Approximate POMDP Planning Toolkit

- APPL contains an efficient C++ implementation of SARSOP
- Supports factored (and non-factored) representations
- Takes as input a POMDP model (in text or XML) and outputs a policy represented as XML
- Source is available (g++ and Visual Studio)
- Includes tools for simulation, evaluation, and visualization

`http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl`

Example input file

```
discount: 0.9
values: reward
states: h0 h1
actions: f0 f1
observations: c0 c1
start: uniform

T: f0 : h0 : h1 0.1
T: f0 : h0 : h0 0.9
T: f0 : h1 : h1 1
T: f1 : * : h0 1

O: * : h0
0.9 0.1

O: * : h1
0.2 0.8

R: f0 : h1 : * : * -10
R: f1 : h0 : * : *  -5
R: f1 : h1 : * : * -15
```

Format specified here: `http://www.pomdp.org/pomdp/code/pomdp-file-spec.shtml`, Matlab code for parsing available here: `http://staff.science.uva.nl/~mtjspaan/software/pomdp`

Example execution

```
Loading the model ...
  input file   : cry.pomdp
  loading time : 0.00s

SARSOP initializing ...
  initialization time : 0.00s

-------------------------------------------------------------------------------
 Time   |#Trial |#Backup |LBound    |UBound    |Precision  |#Alphas |#Beliefs
-------------------------------------------------------------------------------
 0.001   0       0        -55.0001   -22.7678   32.2323     2        1
 0.002   5       51       -25.3301   -24.5726   0.757554    9        6
 0.003   10      105      -24.6886   -24.6415   0.0471187   2        11
 0.004   13      150      -24.6758   -24.6552   0.0205842   5        11
 0.005   17      207      -24.675    -24.6699   0.0050443   2        16
 0.006   20      259      -24.6749   -24.6726   0.00228751  2        18
 0.007   23      305      -24.6749   -24.674    0.000950473 2        18
 0.007   23      305      -24.6749   -24.674    0.000950473 2        18
-------------------------------------------------------------------------------

Writing out policy ...
  output file : out.policy
```

Example output file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Policy version="0.1" type="value" model="cry.pomdp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="policyx.xsd">
<AlphaVector vectorLength="2" numObsValue="1" numVectors="2">
<Vector action="0" obsValue="0">-16.3055 -38.2512 </Vector>
<Vector action="1" obsValue="0">-19.6749 -29.6749 </Vector>
</AlphaVector> </Policy>
```

Format specified here: http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PolicyXDocumentation

Outline

Online methods

- Online methods determine the optimal policy by planning from the current belief state
- Many online methods use a depth-first tree-based search up to some horizon
- The belief states reachable from the current state is typically small compared to the full belief space
- Time complexity is exponential in the search depth
- Heuristics and brand-and-bound techniques allow search space to be pruned

DMU 6.5, S. Ross, J. Pineau, S. Paquet, *et al.*, "Online planning algorithms for POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008

Forward search

```
function $\textsc{SelectAction}(b, d)$
    if $d = 0$
        return $(\textsc{nil}, U(b))$
    $(a^*, u^*) \leftarrow (\textsc{nil}, -\infty)$
    for $a \in A$
        $u \leftarrow R(b, a)$
        for $o \in O$
            $b' \leftarrow \textsc{UpdateBelief}(b, a, o)$
            $(a', u') \leftarrow \textsc{SelectAction}(b', d - 1)$
            $u \leftarrow u + \gamma P(o \mid b, a)u'$
        if $u > u^*$
            $(a^*, u^*) \leftarrow (a, u)$
    return $(a^*, u^*)$
```

Branch and bound

```
function SelectAction($b, d$)
    if $d = 0$
        return $(\textsc{nil}, \underline{U}(b))$
    $(a^*, \underline{u}) \leftarrow (\textsc{nil}, -\infty)$
    for $a \in A$
        if $\overline{U}(b, a) \leq \underline{u}$
            return $(a^*, \underline{u})$
        $u \leftarrow R(b, a)$
        for $o \in O$
            $b' \leftarrow$ UpdateBelief($b, a, o$)
            $(a', \underline{u}') \leftarrow$ SelectAction($b', d - 1$)
            $u \leftarrow u + \gamma P(o \mid b, a)\underline{u}'$
        if $u > \underline{u}$
            $(a^*, \underline{u}) \leftarrow (a, u)$
    return $(a^*, \underline{u})$
```

DMU 6.5.3, Matlab code with pruning (RTBSS) available here: https://cas-robotics.svn.sourceforge.net/svnroot/cas-robotics/pomdpMatlabScripts/commonScripts/

Online methods

- In practice, online methods can be very slow, requiring relatively shallow horizons
- Instead of single-step actions, can use action sequences or control strategies to reduce required depth
- Hybrid methods combine offline and online methods; offline approximate solution provides rough heuristics for leaf nodes in online search
- Online methods are usually insensitive to dimensionality of the state space

DMU 6.5, S. Ross, J. Pineau, S. Paquet, *et al.*, "Online planning algorithms for POMDPs," *Journal of Artificial Intelligence Research*, vol. 32, pp. 663–704, 2008

Some things we didn't talk about

- Partially observable stochastic games (POSGs)
  - See MDPAI 8
- Decentralized POMDPs
  - See MDPAI 9
- Reinforcement learning in POMDPs
  - See S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing Systems*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds., vol. 12, Cambridge, Mass.: MIT Press, 2000, pp. 1064–1070
- Continuous state POMDPs
  - See H. Bai, D. Hsu, W. S. Lee, *et al.*, "Monte Carlo value iteration for continuous state POMDPs," in *International Workshop on the Algorithmic Foundations of Robotics*, 2010
- Mostly observable MDPs
  - See S. C. W. Ong, S. W. Png, D. Hsu, *et al.*, "Planning under uncertainty for robotic tasks with mixed observability," *International Journal of Robotics Research*, vol. 29, no. 8, pp. 1053–1068, 2010