Intelligent Autonomous Agents

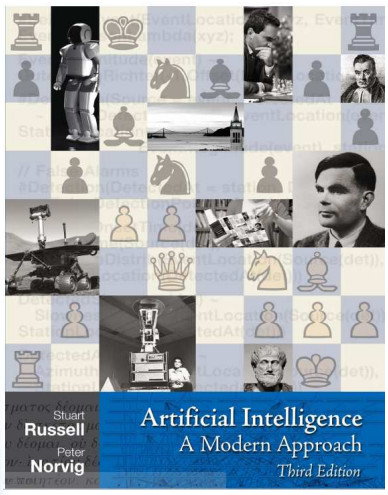
Agents and Rational Behavior

Topic 9: Decision-Making under Uncertainty
Decision-Theoretic Agent Design

Ralf Möller, Rainer Marrone
Hamburg University of Technology

1

Literature



- Chapter 17

2

Last time

- Sequential decision making (uncertain actions)
  - Need a policy -> best action for each possible state
- Finding the best policy
  - Value iteration

repeat
$U \leftarrow U'; \delta \leftarrow 0$
for each state s **in** S **do**
$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s,a)\, U[s']$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s,a) U(s')$$

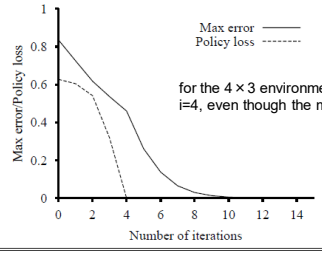
  - Bellman update is a contraction →
    Lead to the definition of when to stop value iteration.

3

Policy Loss

- The policy loss of π_i is connected to the error in U_i *by the following inequality:*

if $||U_i - U|| < \varepsilon$ then $||U^{\pi_i} - U|| < 2\varepsilon$ (17.9)



for the 4 × 3 environment with γ =0.9. The policy πi is optimal when i=4, even though the maximum error in Ui is still 0.46

Figure 17.6 **FILES:** . The maximum error $||U_i - U||$ of the utility estimates and the policy loss $||U^{\pi_i} - U||$, as a function of the number of iterations of value iteration.

4

Last time: Policy iteration

- Create a random policy

  Repeat:
  - Value determination

    $\mathcal{U}_t(i) \leftarrow \mathcal{R}(i) + \sum_k \mathbf{P}(k \mid \Pi(i))\, \mathcal{U}_t(k)$
  - Policy Update:

    $\Pi'(i) = \arg\max_a \sum_k \mathbf{P}(k \mid a.i)\, \mathcal{U}(k)$
  - If $\Pi' = \Pi$ then return Π
- We can combine Value- and Policy Iteration to get the best of both

5

Further optimization

- All algorithms require updating the utility or policy for all states at once.
- At each step we can also select a subset for updating
  asynchronous policy iteration/mod. value iter.
  (can show it will converge if some conditions for initial policy and utility function hold)
- Leads to heuristic algorithms that concentrate on states that are likely to be reached by a good policy.
  - "if one has no intention of throwing oneself off a cliff, one should not spend time worrying about the exact value of the resulting state"

6

Summary

- Decision making under uncertainty
- Sequential decision making
  - Utility of histories
  - Value iteration
  - Policy iteration

7

Jumping-off Point

- Let us assume again that the agent lives in the 4x3 environment
- The agent knows the environment
- BUT
  - Agent has no or very unreliable sensors
  - It does not make sense to determine the optimal policy wrt. a single state
  - $\Pi^*(s)$ is not well defined

8

POMDP: Uncertainty

- Uncertainty about the action outcome
- Uncertainty about the world state due to imperfect (partial) information

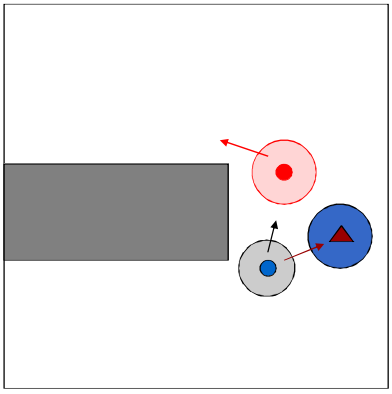
9

Example: Target Tracking

There is uncertainty in the robot's and target's positions; this uncertainty grows with further motion

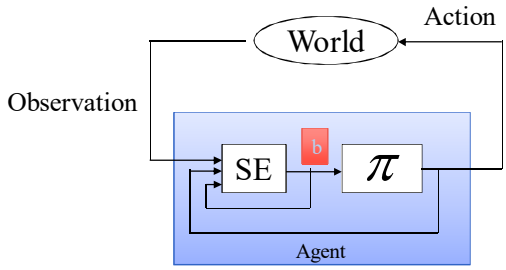
There is a risk that the target may escape behind the corner, requiring the robot to move appropriately

But there is a positioning landmark nearby. Should the robot try to reduce its position uncertainty?



10

Decision cycle of a POMDP agent



- Given the current belief state *b*, execute the action
  $$a = \pi^*(b)$$
- Receive observation *o*
- Set the current belief state to *FORWARD(b,a,o)* and repeat

11

Example Scenario

The agent has no sensors!!!

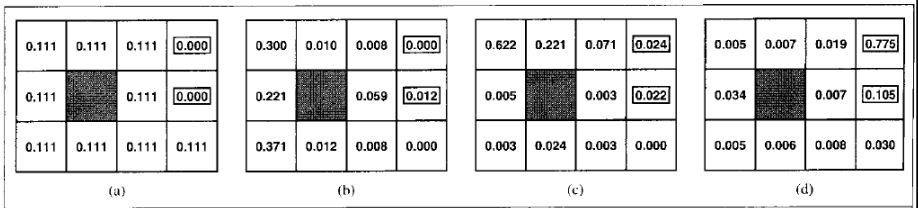


Figure 17.8 (a) The initial probability distribution for the agent's location. (b) After moving *Left* five times. (c) After moving *Up* five times. (d) After moving *Right* five times.

12

Belief state

- *b(s)* is the probability assigned to the actual state *s* by belief state *b*.

0.111	0.111	0.111	0.000
0.111		0.111	0.000
0.111	0.111	0.111	0.111

if **a** is executed in **b** and observation is **e** the belief in **s'** is?

$$\left(\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},\frac{1}{9},0,0\right)$$

$$b'(s') = P(e|s')\sum_{s} P(s'|s,a)b(s) \quad this\ is\ Filtering$$

b' = α *FORWARD(b,a,e)*

13

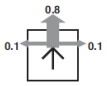
Outcome of actions

- Probability of an observation **e** given that **a** was performed in **b**
  $P(e|a,b) = \sum_{s'} P(e|a,s',b)\, P(s'|a,b)$
  $= \sum_{s'} P(e|s')\, P(s'|a,b)$ markov assumption
  $= \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)\, b(s)$
- Probability of reaching b' from b, given action **a** not knowing **e**
  $P(b'|a,b) = \sum_{e} P(b'|e,a,b)\, P(e|a,b)$
  $= \sum_{e} P(b'|e,a,b) \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)\, b(s)$
  Where P(b'|e,a,b) = 1 if FORWARD(b, a, e) = b' and P(b'|b, a, e) = 0 otherwise
- A new reward function for belief states: $\rho(b)=\sum_{s} b(s)R(s)$
- P(b'|b,a) and $\rho(b)$ define an *observable* MDP on the space of belief states.

14

Belief MDP

- A belief MDP is a tuple <B, A, ρ, E>:
  B = infinite set of belief states
  E = percepts
  A = finite set of actions
  $\rho(b) = \sum_s b(s)R(s)$ (reward function)



$P(b'|b, a) = \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a)\, b(s)$ (transition function)

Move left once, without observations

b ≈ b'

0.111	0.111	0.111	0.000
0.111		0.111	0.000
0.111	0.111	0.111	0.111

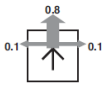
? 0.2

0.8*0.111+0.1*0.111 +0.1*0.111 +0.8*0.111 = 0.2

15

Belief MDP

- A belief MDP is a tuple <B, A, ρ, E>:
  B = infinite set of belief states
  E = percepts
  A = finite set of actions
  $\rho(b) = \sum_s b(s)R(s)$ (reward function)



$P(b'|b, a) = \sum_e P(b'|e,a,b) \sum_{s'} P(e|s') \sum_s P(s'|s,a)\, b(s)$ (transition function)

Move left once, without observations

b ≈ b'

0.111	0.111	0.111	0.000
0.111		0.111	0.000
0.111	0.111	0.111	0.111

0.2

? 0.111

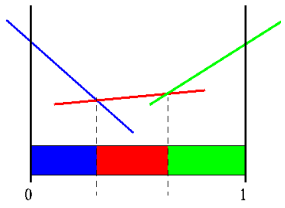
0.8*0.111 +0.1*0.111 +0.1*0.111 = 0.111

16

Solutions for POMDP

- Methods based on *value* and *policy iteration*:

  A policy $\pi(b)$ can be represented as a set of *regions* of belief state space, each of which is associated with a particular optimal action. The value function associates a distinct *linear* function of *b* with each region. Each value or policy iteration step refines the boundaries of the regions and may introduce new regions.



18

Value Iteration for POMDPS

- Consider an optimal policy π* and its application in **belief state b**.
- For this **b** the policy is a "conditional plan"
  - Let the utility of executing a fixed conditional plan **p** in **s** be $u_p(s)$.
    Expected utility $U_p(b) = \sum_s b(s)\, u_p(s)$
    It varies linearly with b, a hyperplane in a belief space
  - At any **b**, the optimal policy will choose the conditional plan with the highest expected utility
    $U(b) = U^{\pi^*}(b) = \text{argmax}_p\ b \times u_p$ (summation of dot-prod.)
- U(b) is the maximum of a collection of hyperplanes and will be piecewise linear and convex

19

Example: Conditional Plans

- Two state world 0,1
- Two actions: stay(s), go(s)
  - Actions achieve intended effect with some probability p
- One-step plan [go], [stay]
- Two-step plans are conditional
  - [a1, IF percept = 0 THEN a2 ELSE a3]
  - Shorthand notation: [a1, a2/a3]
- n-step plan are trees with nodes attached with actions and edges attached with percepts

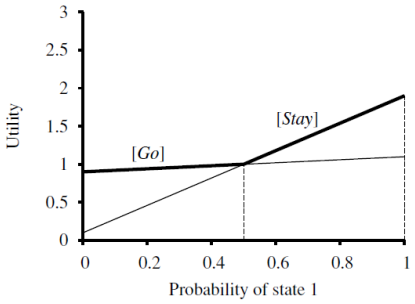
20

Example

- Two state world 0,1. R(0)=0, R(1)=1
- Two actions: stay (0.9), go (0.9)
- The sensor reports the correct state with prob. 0.6
- Consider the one-step plans [stay] and [go]
  - $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
  - $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
  - $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
  - $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$
- This is just the direct reward function (taken into account the probabilistic transitions)

21

Example



Utility of two one-step plans as a function of b(1)

if(b(1)>0.5) stay else go

22

General formula

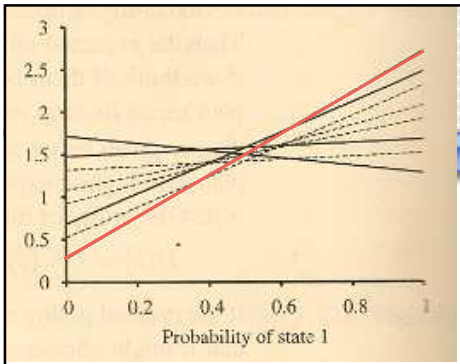
We can compute the utilities for conditional plans of depth-2 by considering each possible first action, each possible subsequent percept and then each way of choosing a depth-1 plan to execute for each percept

[Stay; **if** Percept =0 **then** Stay **else** Stay]
[Stay; **if** Percept =0 **then** Stay **else** Go] . . .

- Let *p* be a depth-*d* conditional plan whose initial action is ***a*** and whose depth-*(d-1)* subplan for percept **e** is ***p.e***, then

$u_p(s) = R(s) + \sum_{s'} P(s'|\, s,a) \sum_e P(e|s')\, u_{p.e}(s')$

23



Example

- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

$$u_p(s) = R(s) + \sum_{s'} P(s' | s,a) \sum_e P(e|s') u_{p.e}(s')$$

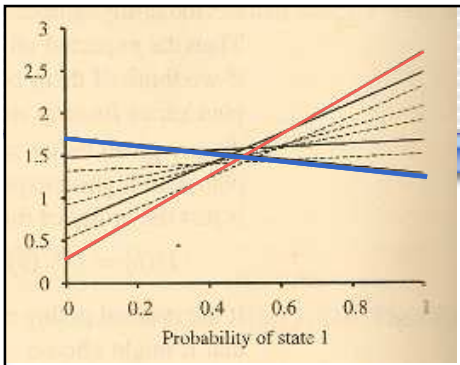
$u_{[stay,\ stay/stay]}(0)=R(0) + (0.9*(0.6*0.1 + 0.4*0.1) + 0.1*(0.4*1.9 + 0.6*1.9))=0.28$

use $u_{stay}(0)$ use $u_{stay}(1)$

$u_{[stay,\ stay/stay]}(1)=R(1) + (0.1*(0.6*0.1 + 0.4*0.1)+ 0.9*(0.4*1.9 + 0.6*1.9))=2.72$

$u_{stay}(0)$ $u_{stay}(1)$

24



Example

- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

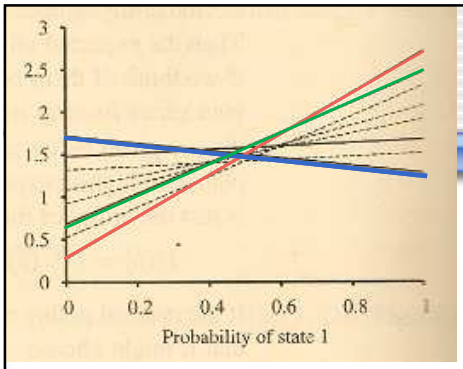
$$u_p(s) = R(s) + \sum_{s'} P(s' | s,a) \sum_e P(e|s') u_{p.e}(s')$$

$u_{[go,\ stay/stay]}(0)=R(0) + (0.1*(0.6*0.1 + 0.4*0.1) + 0.9*(0.4*1.9 + 0.6*1.9))=1.72$

$u_{[go,\ stay/stay]}(1)=R(1) + (0.9*(0.6*0.1 + 0.4*0.1) + 0.1*(0.4*1.9 + 0.6*1.9))=1.28$

$u_{stay}(0)$ $u_{stay}(1)$

25



Example

- $u_{[stay]}(0)=R(0) + 0.9R(0)+0.1R(1) = 0.1$
- $u_{[stay]}(1)=R(1) + 0.9R(1)+0.1R(0) = 1.9$
- $u_{[go]}(0)=R(0) + 0.9R(1)+0.1R(0) = 0.9$
- $u_{[go]}(1)=R(1) + 0.9R(0)+0.1R(1) = 1.1$

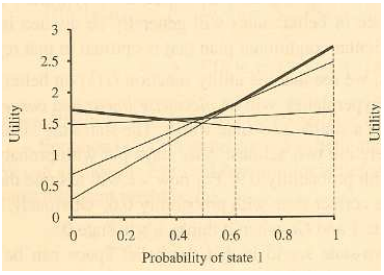
$$u_p(s) = R(s) + \sum_{s'} P(s' | s,a) \sum_e P(e|s') u_{p.e}(s')$$

$u_{[stay,\ go/stay]}(0)= R(0) + (0.9*(0.6*0.9 + 0.4*0.1) + 0.1*(0.4*1.1 + 0.6*1.9))=0.68$
$u_{go}(0)$ $u_{stay}(0)$ $u_{go}(1)$ $u_{stay}(1)$

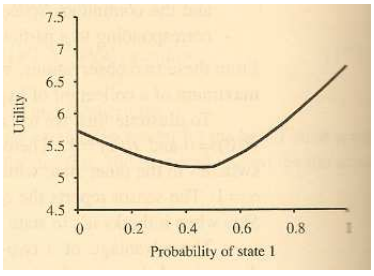
$u_{[stay,\ go/stay]}(1)= R(1) + (0.1*(0.6*0.9 + 0.4*0.1) + 0.9*(0.4*1.1+ 0.6*1.9))=2.48$

26

Example



Utility of four undominated two-step plans



Utility function for optimal eight step plans

27

Value Iteration

$u_p(s) = R(s) + \sum_{s'} P(s' | s,a) \sum_e P(e|s') u_{p.e}(s')$

- This give us a value iteration algorithm
- The elimination of dominated plans is essential for reducing doubly exponential growth:
  the number of undominated plans with d=8 is just 144,
  otherwise 2^{255} $(|A|^{O(|E|^{d-1})})$
  If you have **n** undominated plans you have to generate $|A| * n^{|E|}$ new plans.
- For large POMDPs this approach is highly inefficient

28

Model for POMDPs

- Dynamic Bayesian network
  - the transition and observation models
- Dynamic decision network (DDN)
  - decision and utility
- A filtering algorithm
  - incorporate each new percept and action and update the belief state representation.
- Decisions are made by projecting forward possible action sequences and choosing the best action sequence.

29

The Generic Structure of a Dynamic Decision Network

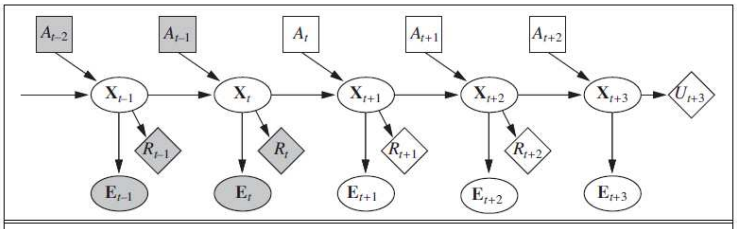
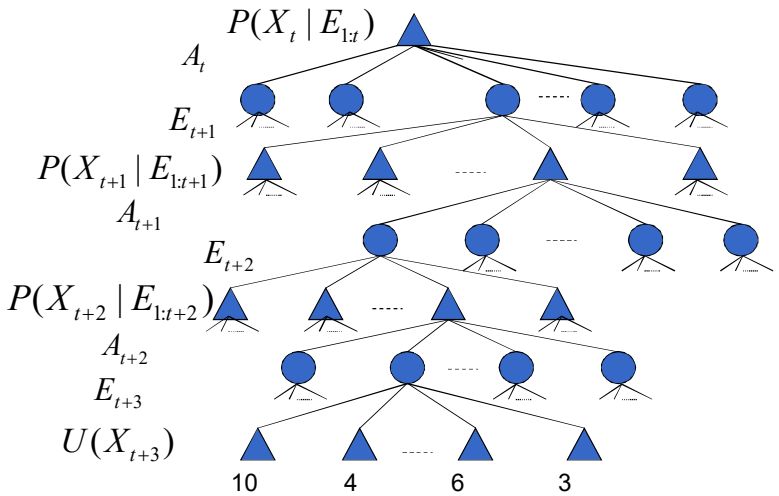


Figure 17.10 **FILES: figures/generic-ddn.eps (Tue Nov 3 16:22:53 2009).** The generic structure of a dynamic decision network. Variables with known values are shaded. The current time is t and the agent must decide what to do—that is, choose a value for A_t. The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon.

- The decision problem involves calculating the value of A_t that maximizes the agent's expected utility over the remaining state sequence.

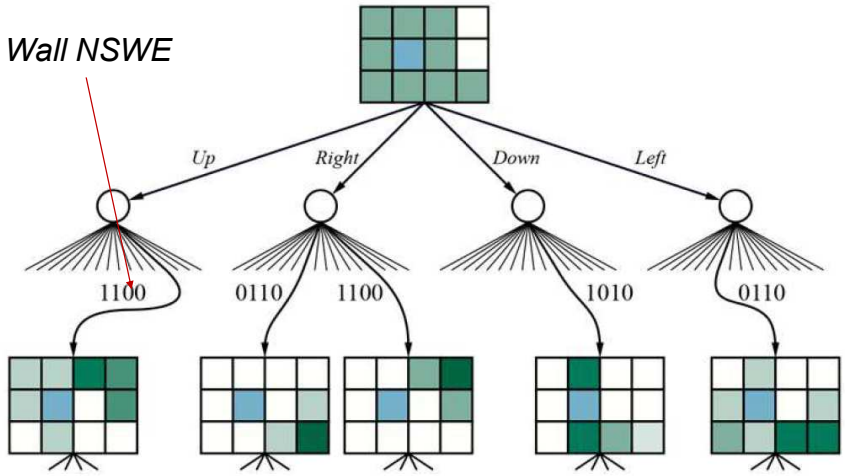
30

Search Tree of the Lookahead DDN



31

Search Tree of the Lookahead DDN



32

Search Tree: Exhaustive Enumeration

- The search tree of DDN is very similar to the EXPECTIMINIMAX algorithm for game trees with chance nodes, except hat:
  - There can also be rewards at non-leaf states
  - The decision nodes correspond to belief states rather than actual states.
- The time complexity: $O(|A|^d \bullet |E|^d)$

  d is the depth, |A| is the number of available actions, |E| is the number of possible observations.
  This is far less than value iteration.

33

Discussion of DDNs

- DDNs provide a **general**, **concise representation** for large POMDPs
- Agent systems moved from
  - **static**, **accessible**, and **simple** environments to
  - **dynamic**, **inaccessible**, and **complex** environments that are closer to the real world
- However, **exact algorithms** are **exponential**

34

Perspectives of DDNs to Reduce Complexity

- **Heuristic estimate** for the utility of the remaining steps
- Incremental **pruning** techniques
- Many **approximation** techniques as in our search lecture:
  - Using less detailed state variables for states in the distant future.
  - Using a greedy heuristic search through the space of decision sequences.

…

35