Artificial Intelligence 2

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Integration

Probability theory describes what an agent should believe on the basis of evidence.

Utility theory describes what an agent wants.

Decision theory puts the two together to describe what an agent should do.

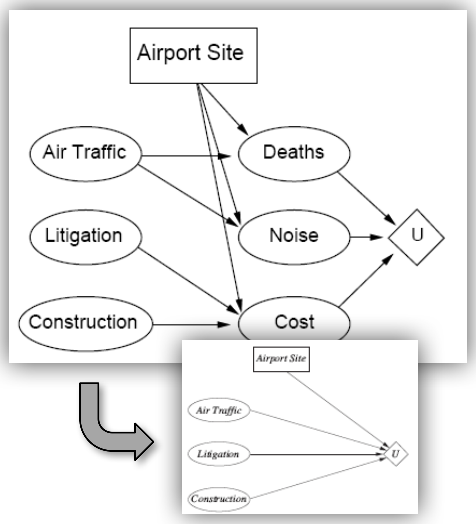
What is next?

- making simple (single) decisions
- sequential decision problems

Decision networks

How to describe a general mechanism of **making rational decisions**?

A **decision network** (influence diagram) represents information about the agent‘s current state, its possible actions, the state that will result from the agent‘s action, and the utility of the state.



Chance nodes (ovals) represent random variables just as in Bayesian networks.

Decision nodes (rectangles) represent points where the decision maker has a choice of actions (a single decision node now).

Utility nodes (diamonds) represent the agent‘s utility function.

Evaluating decision networks

Actions are selected by evaluating the decision network for each possible setting of the decision node.

1. set the evidence variables for the current state
2. for each possible value of the decision node
   a) set the decision node to that value
   b) calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference
   c) calculate the resulting utility for the action
3. return the action with the highest utility

For problems with more utility nodes we will use **multi-attribute utility theory**.

Constructing decision networks

Create a causal model

determine the possible symptoms, disorders, treatments, and outcomes and then draw arcs between them

Simplify to a qualitative decision model

we can simplify by removing variables that are not involved in treatment decisions; sometime variables will have to be split or joined to match the expert's intuitions

Assign probabilities

fill CPTs in the Bayesian networks
(from patient databases, literature studies
or expert's subjective assessments)

Assign utilities

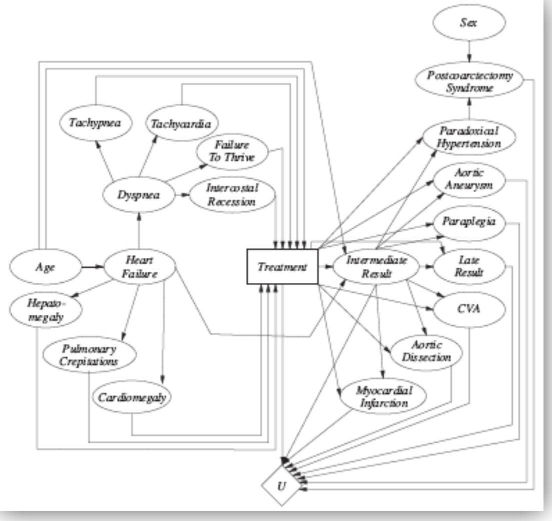
a small number of possible outcomes can be enumerated (can be done by the expert , but better if the patient is involved)

Verify and refine the model

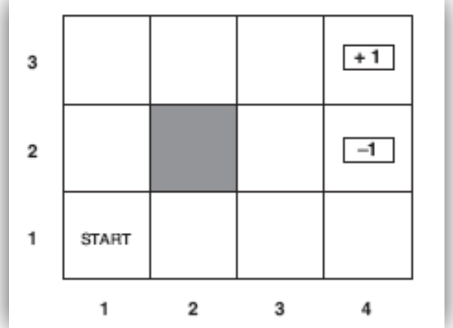
compare outputs with a so-called gold standard (a team of best doctors)

Perform sensitivity analysis

check whether the best decision is sensitive to small changes in the assigned probabilities and utilities by systematically varying those parameters and running the evaluation again (small changes leading to significantly different decisions indicate problems)



Sequential decision problems

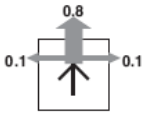


Suppose that an agent is situated in the 3 x 4 environment.

The environment is **fully observable** so that the agent always knows where it is

The actions in every state are Up, Down, Left, Right

- the actions available in state s are denoted A(s)
- actions are **unreliable**
  - each action achieves the intended effect with probability 0.8
  - the rest of time, the action moves the agent at right angles to the intended direction
- probability that [Up,Up,Right,Right,Right] reaches the goal (+1) from START is $0.8^5 + 0.1^4 \text{ x } 0.8 = 0.32776$.



Markov decision process

The **transition model** describes the outcome of each action in each state using the classical approach P(s‘|s,a) – probability of reaching state *s‘* if action *a* is done in state *s*.

- we will assume that transitions are **Markovian** (probability of reaching *s‘* from *s* does not depend on the history of earlier states)

The **utility function** will depend on a sequence of states.

- each state has a (bounded) **reward** R(s)
  - Example: +1 (goal exit), -1 (unwanted exit), -0.04 (other states)
- **utility function** is (for now) the sum of the rewards received at visited states
  - it is similar to looking for a shortest path to the goal in a stochastic environment

Markov Decision Process (MDP)

a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards

Solution to an MDP

A fixed sequence of actions (a plan) cannot be used in stochastic environments

- agent might end up in a state other than the goal

A solution must specify what the agent should do for any state that the agent might reach.

A solution to an MDP is a policy – a function recommending an action for each state – π(s)

- an **optimal policy** is a policy that yields the highest expected utility
- this policy represents the agent function explicitly and it is therefore a description of a **simple reflex agent**

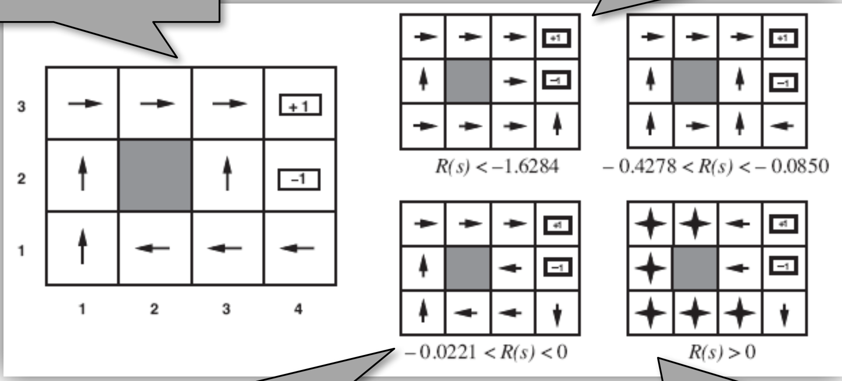
Optimal policies

optimal policy yields the highest expected utility

– it depends on particular values of rewards

Reward of states is -0.04
The agent is heading for the goal exit but is conservative

Reward of states is negative
Life is so painful that the agent heads straight for the nearest exist



$R(s) < -1.6284$

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

$-0.0221 < R(s) < 0$

$R(s) > 0$

Reward of states is close to 0
The agent is heading the goal exit but takes no risks at all

Reward of states is positive
Life is positively enjoyable and the agent avoids both exits

Utilities over time (horizon)

How is the utility function over a sequence of states defined?

Similarly to **multi-attribute utility function** $U([s_0,s_1,\ldots,s_n])$ (states correspond to attributes), but

what is the horizon?

- **finite horizon**
  - there is a fixed time N after which nothing matters
    $U([s_0,s_1,\ldots,s_{N+k}]) = U([s_0,s_1,\ldots,s_N])$
  - The optimal action in a given state could change over time (optimal policy is nonstationary)
    - for N=3 and state (3,1) the agents selects the action Up
    - for N=100 and state (3,1) the agent selects the action Left
- **infinite horizon**
  - there is no reason to behave differently in the same state at different times
  - **the optimal policy is stationary**
  - infinite horizon does not necessarily mean that all state sequences are infinite; it just means that there is no fixed deadline

Utilities over time

Utility function behaves as an **multiattribute utility function** $U([s_0,s_1,\ldots,s_n])$.

To obtain a simple expression we assume that agent's preferences between state sequences are **stationary**

- $[s_0,s_1,s_2,\ldots]$ and $[s_0,s'_1,s'_2,\ldots]$ are preference-ordered the same way as the sequences $[s_1,s_2,\ldots]$ and $[s'_1,s'_2,\ldots]$
- the agent's preference „tomorrow and today" are identical

Under stationarity there are just two coherent ways to assign utilities to sequences:

- **additive rewards**

  $U([s_0,s_1,s_2,\ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
- **discounted rewards**

  $U([s_0,s_1,s_2,\ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$

  discount factor $\gamma \in (0,1)$
  - describes the preference of an agent for current rewards over future rewards (close to 0 = rewards in the distant future are viewed as insignificant; 1 = discounted rewards are exactly equivalent to additive rewards)
  - equivalent to an interest rate $1/\gamma - 1$

Utilities over time (properties)

In the reminder, we assume **discounted rewards.**

- If the environment does not contain a terminal state (or if the agent never reaches one) and utilities are additive then undiscounted rewards will generally be $+\infty$ or $-\infty$
- With discounted rewards the **utility** of an infinite sequence is **finite** (let rewards are bounded by $\pm R_{max}$)

  $$U([s_0,s_1,s_2,\ldots]) = \Sigma_{i=0,\ldots,+\infty} \gamma^i R(s_i) \leq \Sigma_{i=0,\ldots,+\infty} \gamma^i R_{max} = R_{max} / (1-\gamma)$$
- If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.
  - A policy that is guaranteed to reach a terminal state is called a **proper policy**
    - we can use additive rewards
- Infinite sequences can be compared in terms of the **average reward** obtained per time step (better to stay in state with a reward 0.1 than in a state with a reward 0.01).

Optimal policies

The expected utility obtained by executing π starting in *s* is given by:

$U^{\pi}(s) = E[\Sigma_{i=0,\dots,+\infty} \gamma^{i} R(S_i)]$

- S_t is a random variable describing the state that the agent reaches at time *t*

Optimal policy for the initial state s is defined as:

$\pi^*_s = \text{argmax}_{\pi} U^{\pi}(s)$

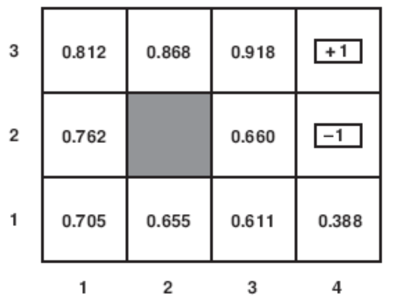
Does the optimal policy depend on the initial state?

- if two policies π^*_a and π^*_b reach a third state *c*, there is no good reason for them to disagree with each other about what to do next

Let us define the **true utility of a state** as just $U(s) = U^{\pi^*}(s)$

- R(s) is the "short term" reward for being in *s*, whereas U(s) is the "long term" total reward from *s* onward
- Then choose the action that maximizes the expected utility of the subsequent state

  $\pi^*(s) = \text{argmax}_{a \in A(s)} \Sigma_{s'} P(s'|s,a) U(s')$
- Notice that this does not need to be the action going to the state with largest U(s)!



3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

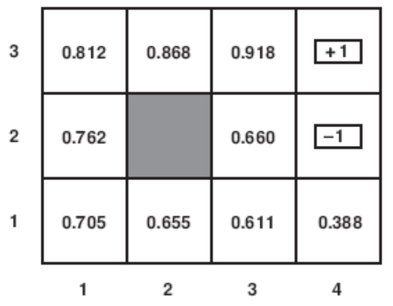
Bellman equation

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \Sigma_{s'} P(s'|s,a) U(s')$$

This is called the **Bellman equation**.

$U((1,1)) = -0.04 +$
$\gamma \max[$
$0.8U((1,2)) + 0.1U((2,1)) + 0.1U((1,1)),$
$0.9U((1,1)) + 0.1U((1,2)),$
$0.9U((1,1)) + 0.1U((2,1)),$
$0.8U((2,1)) + 0.1U((1,2)) + 0.1U((1,1))]$



3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Value iteration

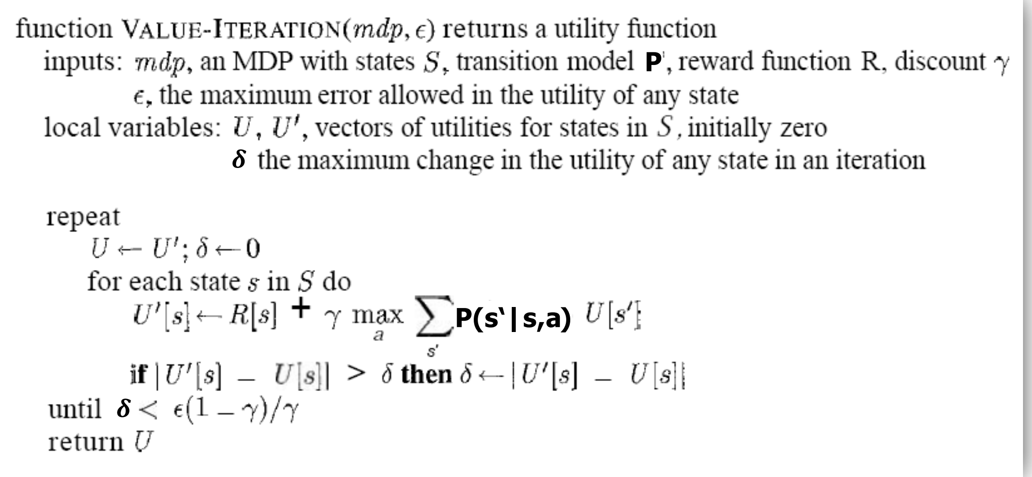
The Bellman equation is the basis of the **value iteration algorithm** for solving MDPs.

There is one problem: the equations are **nonlinear**.

We will apply an **iterative approach**

1. We start with arbitrary initial values for the utilities U(s)
2. We update the utility U(s) of each state from the utilities of its neighbors (a **Bellman update**)

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a\in A(s)} \Sigma_{s'} P(s'|s,a)\, U(s')$$



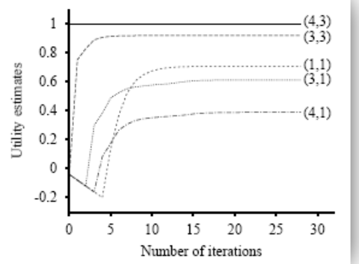
```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, transition model P', reward function R, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$ the maximum change in the utility of any state in an iteration

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

Convergence of value iteration

How do we know that the value iteration algorithm eventually converges to a unique set of solutions of the Bellman equations?

- The basic concept is the notion of **contraction** ($|f(x) - f(x')| < c|x - x'|$, where $0 \leq c < 1$)
  - function f has only one fixed point
  - when the function is applied to any argument, the value must get closer to fixed point
- Let U be a vector of utilities for all the states, we will use the **max norm** to measure "length" of a vector.
  $|U| = \max_s |U(s)|$
- Let BU be the Bellman update of vector U, then
  $|BU_i - BU'_i| \leq \gamma\ |U_i - U'_i|$
  $|BU_i - U| \leq \gamma\ |U_i - U|$, where U is the vector of true utilities (the fixed point)



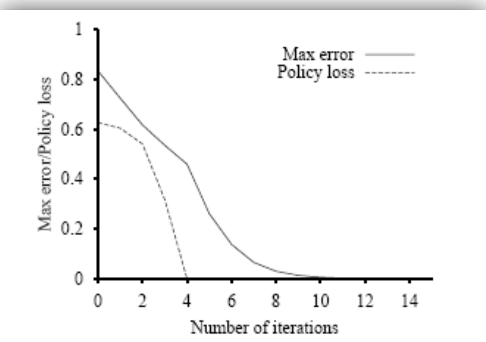
- **The number of iterations** to reach an error of at most ε
  $|BU_0 - U| \leq 2R_{max}/(1-\gamma)$ the maximal distance from the true utility
  $|BU_N - U| \leq \gamma^N 2R_{max}/(1-\gamma)$ in each iteration, the error is reduces by at least γ
  $N = \lceil \log(2R_{max}/\varepsilon(1-\gamma))/\log(1/\gamma) \rceil$ #iterations to reach an error of at most ε
- **The terminal condition** to reach an error of at most ε from the true utility U
  $|BU_{i+1} - BU_i| < \varepsilon\,(1-\gamma)/\gamma$

Policy loss

What the agent really cares about is how well it will do if it makes its decisions on the basis of the current utility function U_i.

- **policy loss** is the most the agent can lose by executing π, instead of the optimal policy π^*: $|U^{\pi_i} - U|$
- if $|U_i - U| < \varepsilon$ then $|U^{\pi_i} - U| < 2\varepsilon\,\gamma\,/\,(1 - \gamma)$

In practice, it often occurs that π_i becomes optimal long before U_i has converged.



Policy iteration

It is possible to get an optimal policy even when the utility function estimate is inaccurate.

- If one action is clearly better than all others, then the exact magnitude of the utilities on the states involved need not be precise.

This suggests an alternative way to find optimal policies – **the policy iteration algorithm** that alternates the following two steps:

- **policy evaluation** – given a policy π_i, calculate U^{π_i}
  - the action in each state is fixed by the policy so we have a simplified version of the Bellman equations

    $$U^{\pi_i}(s) = R(s) + \gamma\, \Sigma_{s'}\, P(s'|s, \pi_i(s))\, U^{\pi_i}(s')$$

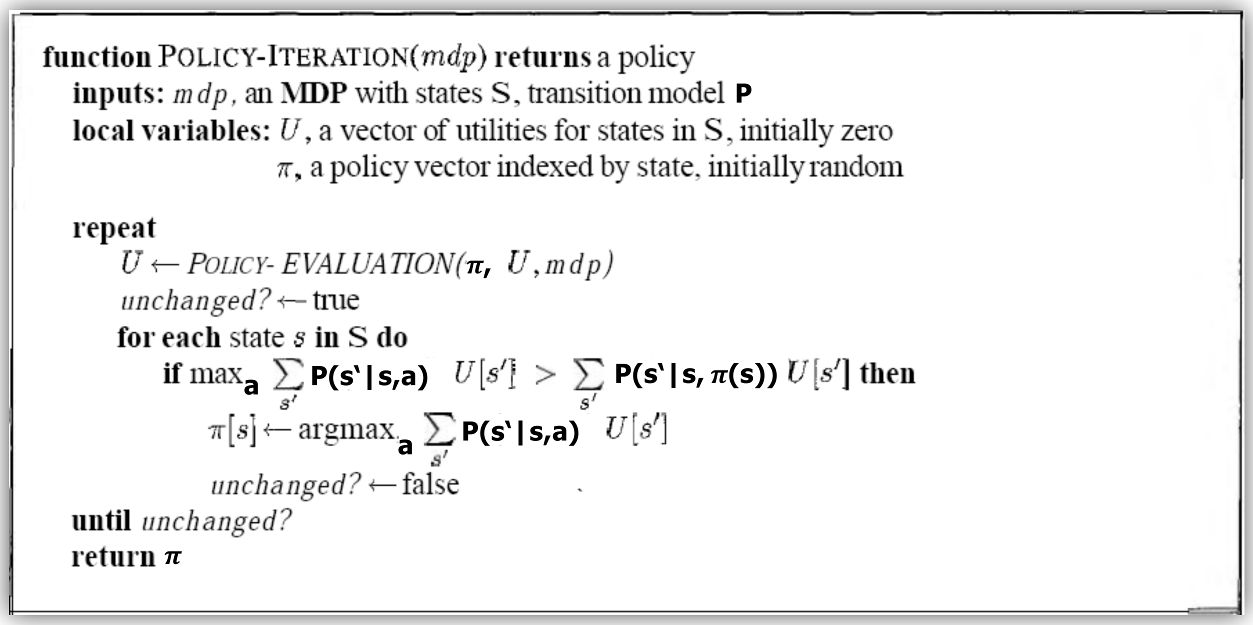
  - for small state spaces, policy evaluation using exact solution methods is often the most efficient approach – time comlexity is $O(n^3)$
  - for large state spaces, we can perform some number of simplified value iteration steps to give a reasonably good approximation of the utilities

    $$U_{j+1}(s) \leftarrow R(s) + \gamma\, \Sigma_{s'}\, P(s'|s, \pi_i(s))\, U_j(s')$$

- **policy improvement**

  $$\pi_{i+1}(s) \leftarrow \mathrm{argmax}_{a \in A(s)}\, \Sigma_{s'}\, P(s'|s,a)\, U^{\pi_i}(s')$$

Policy iteration



```
function POLICY-ITERATION(mdp) returns a policy
   inputs: mdp, an MDP with states S, transition model P
   local variables: U, a vector of utilities for states in S, initially zero
                    π, a policy vector indexed by state, initially random

   repeat
       U ← POLICY-EVALUATION(π, U, mdp)
       unchanged? ← true
       for each state s in S do
           if $\max_a \sum_{s'} P(s`|s,a)\ U[s'] > \sum_{s'} P(s`|s,\pi(s))\ U[s']$ then
               $\pi[s] \leftarrow \operatorname{argmax}_a \sum_{s'} P(s`|s,a)\ U[s']$
               unchanged? ← false
   until unchanged?
   return π
```

Because there are only finitely many policies for a finite state space, and each iteration yields a better policy, **policy iteration must terminate**.

Next?

For **fully observable environment** (agent knows the current state) we can suggest the action (**policy**) maximizing the expected utility even if the **output of the action is uncertain**.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \Sigma_{s'} P(s'|s,a)\ U(s')$$

What if the environment is only partially observable?

- partially observable Markov decision process (POMDP)

POMDP

A POMDP has the same elements as an MDP:

- the transition model P(s‘|s,a)
- reward function R(s)
- sensor model P(e|s)

The agent does not necessarily know which state it is in, so it cannot execute the action π(s) recommended for that state.

How to model an unknown state?

- a **belief state**
- a probability distribution over all possible states (modelled as a vector of probabilities of each state)
- b(s) = probability of being in state *s* in the belief state *b*

We will look for an optimal policy for belief states π(b).

Solving POMDP

How to calculate a belief state b‘ after applying action *a* to a belief state *b*?

$b‘(s‘) = \alpha\ P(e|s‘)\ \Sigma_s\ P(s‘|s,a)\ b(s)$

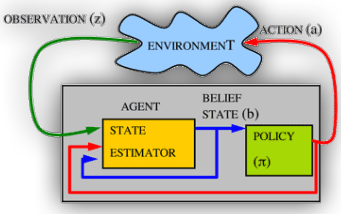
b‘ = FORWARD(b,a,e)

- This is essentially the **filtering** task and it can be solved incrementally.

The optimal action depends only on the agent’s current belief state π*(b).

The decision cycle of a **POMDP agent** can be broken down into the following three steps:

1. given the current belief state b, **execute the action** a = π*(b)
2. receive **percept** e.
3. set the current **belief state** b‘ = FORWARD(b,a,e) and repeat.



From POMDP to MDP

Notice that the next belief state b‘ is calculated deterministically given the current belief state b, action a, and subsequent percept e.

$b' = \text{FORWARD}(b,a,e)$

How to calculate the next belief state when the observation is not yet available?

- The probability of perceiving *e*, given that *a* was performed in belief state *b*:
  $$P(e|a,b) = \Sigma_{s'} P(e|a,b,s') P(s'|a,b)$$
  $$= \Sigma_{s'} P(e|s') P(s'|a,b)$$
  $$= \Sigma_{s'} P(e|s') \Sigma_s P(s'|a,s) b(s)$$
- The probability of reaching b’ from b, given an action s:
  $$P(b'|b,a) = P(b'|a,b) = \Sigma_e P(b'|a,b,e) P(e|a,b)$$
  $$= \Sigma_e P(b'|a,b,e) \Sigma_{s'} P(e|s') \Sigma_s P(s'|a,s) b(s)$$
  where $P(b'|a,b,e) = 1$, if $b' = \text{FORWARD}(b,a,e)$, otherwise 0

The **reward function for belief states** can be defined as

$$r(b) = \Sigma_s b(s)R(s)$$

Together P(b‘|b,a) and r(b) define an **observable MDP!**

- Solving a POMDP on a physical state space can be reduced to solving and MDP on the corresponding belief-state space.
- However, the MDP we obtain has a continuous (and usually multi-dimensional) state space!

Online agents for POMDP

The presented techniques for MDP cannot be directly applied due to infinitely many number of states.

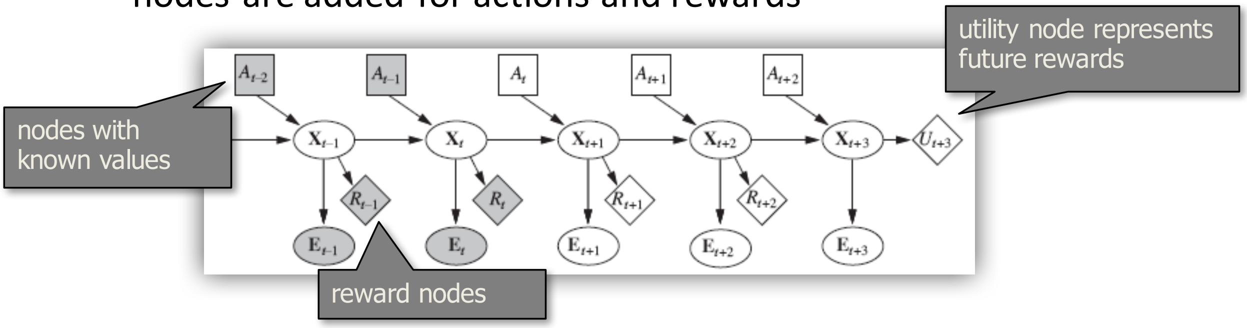
Instead of policies we can use **conditional plans**.

- an action is selected based on observations and previous actions.

Value iteration can be modified for POMDPs but it is not very efficient.

We will use **dynamic Bayesian networks** and look-ahead techniques.

- dynamic Bayesian network represents the transition and sensor models
- nodes are added for actions and rewards



utility node represents future rewards

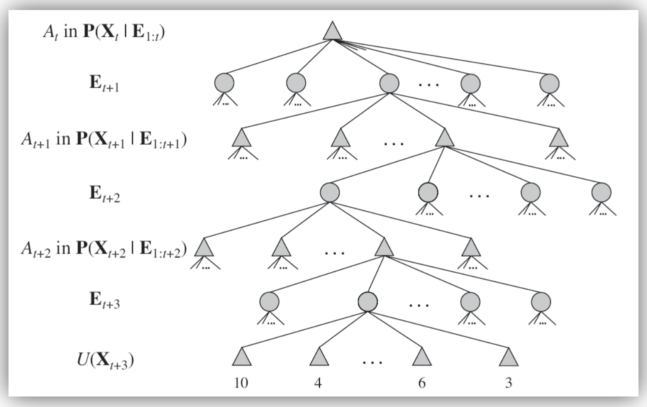
nodes with known values

reward nodes

Look-ahead techniques

The solving techniques are similar to methods of game playing.

- the opponent is equivalent to observations


A_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$
$\mathbf{E}_{t+1}$
A_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$
$\mathbf{E}_{t+2}$
A_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$
$\mathbf{E}_{t+3}$
$U(\mathbf{X}_{t+3})$
10
4
...
6
3

- the triangular nodes represent **belief states** and there are arcs annotated by possible actions
  - belief state is obtained deterministically from the path (actions and observations are known)
- the round (chance) nodes correspond to choices by the environment, namely what **evidence** arrives
- no special nodes for uncertain action effects are necessary as application of action to a belief state gives the next belief state deterministically

A decision can be extracted from the search tree by the algorithm similar to the **EXPECTEDMINIMAX algorithm** for game trees with chance nodes.

The depth of search tree can be determined by the discount factor (the future utility value is small due to discounting).

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz