Introduction to Artificial Intelligence

Roman Barták
Department of Theoretical Computer Science and Mathematical Logic

Recall, that a **rational agent** is an agent that can make rational decisions based on what it believes and what it wants.

Probability theory gives a formal tool to reason about uncertainty of the world.

Now, how can we **make decisions** there?

- we need to measure **outcome quality**
- we will do **simple decisions** (episodic environments)
- we will do a **sequence of decisions** (sequential environments)

Decision theory (simplest form) is about choosing among actions based on desirability of immediate outcomes (environment is episodic).

Previously, **Result(s,a)** denotes the state that is the deterministic outcome of taking action a in state s.

Assume now **nondeterministic partially observable environment** – we do not know the current state and we do not know the outcome of action.

Formal model:

- Result(a), random variable with values describing possible outcome state
- P(Result(a)=s | a, **e**), probability of outcome **s** of action **a**, given evidence observation **e**

Agent's preferences are captured by a **utility function** U(s) – number expressing desirability of a state s.

Expected utility (EU) of an action **a** given the evidence **e**

$$EU(a|\mathbf{e}) = \Sigma_s P(Result(a)=s \mid a,\mathbf{e})U(s)$$

Maximum expected utility (MEU) principle:

$$action = argmax_a EU(a|\mathbf{e})$$

A rational agent should choose the action that maximizes the agent's expected utility.

Frequently, it is easier for an agent to express **preferences** between states rather than to give a number describing the utility value:

- A > B: the agent prefers A over B
- A ~ B: the agent is indifferent between A and B

We can describe outcome of each action as an **lottery** $L = [p_1,S_1; …; p_n,S_n]$ (possible outcomes $S_1, …, S_n$ that occur with probabilities $p_1, …, p_n$)

Example: choice of chicken in airplane: [0.8, juicy; 0.2, overcooked]

Expected utility of a lottery: $U([p_1,S_1; …; p_n,S_n]) = \sum_i p_i U(S_i)$

How to go **from preferences to the utility** function such that:

$U(A) < U(B) \Leftrightarrow A < B$

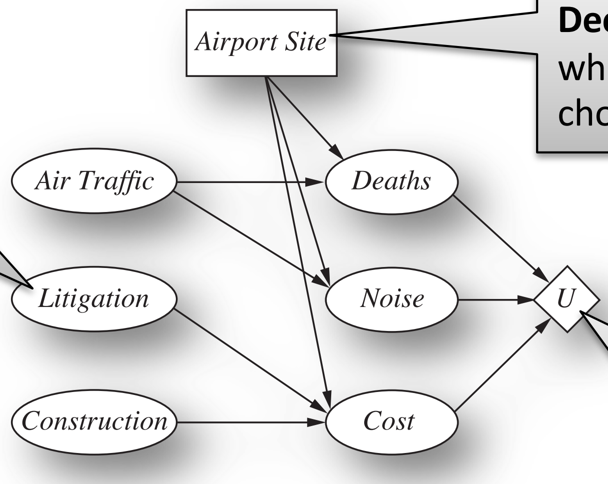
$U(A) = U(B) \Leftrightarrow A \sim B$

We can look for a **normalized utility function** (values between 0 and 1):

- We fix the utility of a "best possible prize" S_{max} to 1, $U(S_{max}) = 1$.
- Similarly, a "worst possible catastrophe" S_{min} is mapped to 0, $U(S_{min}) = 0$.
- Now, to assess the utility of any particular prize S we ask the agent to choose between S and a **standard lottery** $[p, S_{max}; 1-p, S_{min}]$
- The probability p is adjusted until the agent is indifferent between S and the standard lottery.
- Then the utility of S is given by, $U(S) = p$.

Decision networks (influence diagrams) combine Bayesian networks with additional node types for actions and utilities.

Chance nodes represent random variables like in Bayesian networks



Decision node represent points where the decision maker has a choice of actions

Utility nodes represent the agent's utility function

Evaluating decision networks:

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

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

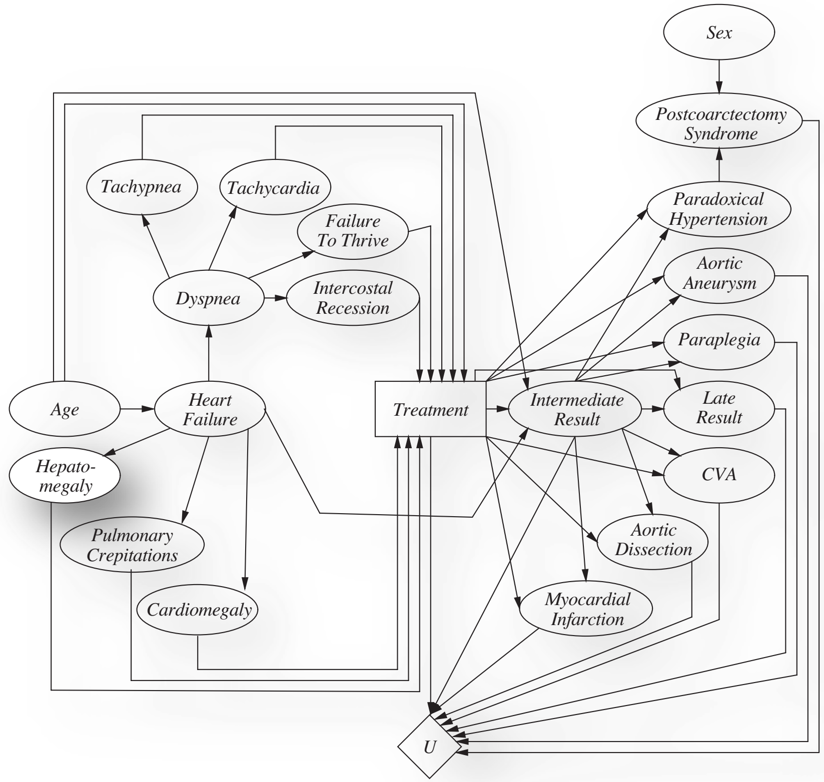
a small number of possible outcomes can be enumerated (can be done by the expert , but better if the patient is involved)

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)



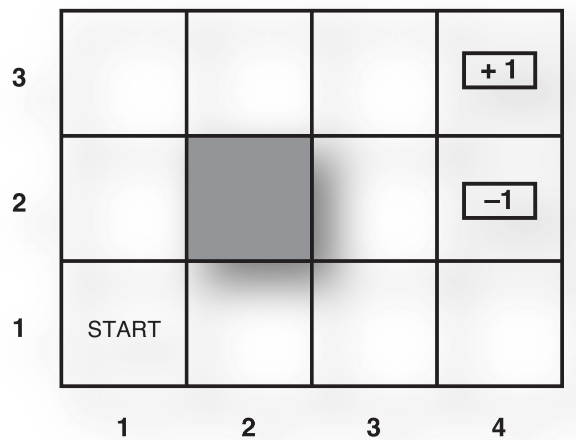
What if we need to decide today, tomorrow, and so on and utility depends on a sequence of decisions?

→ **sequential decision problems**

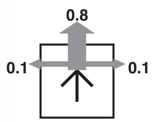
Previously, we used **search** and **planning** as special cases of sequential decision problems, but both assume fully observable deterministic environments.

Let us look at **fully observable** (agent knows where it is) but **non-deterministic environments**.

Example problem:



– an agent is situated in the fully-observable 3 x 4 environment
– actions in every state are Up, Down, Left, Right but their outcome is stochastic



  – 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

Plan [Up,Up,Right,Right,Right] reaches the goal (+1) from START with probability:

$$0.8^5 + 0.1^4 \times 0.8 = 0.32776.$$

Markov Decision Process (MDP) is a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive reward.

Transition model $P(s'|s,a)$ – probability of reaching state *s'* if action *a* is applied in state *s*

Markovian property – probability of reaching *s'* from *s* does not depend on the history of earlier states

Reward $R(s)$ received by an agent at state s

- can be positive or negative, but must be bounded
- it is a "**short term**" reward

Utility function $U([s_0,s_1,s_2,\ldots])$

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

where γ – a **discount factor** – is a number between 0 and 1

- discounted rewards mean that future rewards are less significant
- the utility based on discounted rewards is finite even for an infinite sequence of states ($U([s_0,s_1,s_2,\ldots]) \leq R_{max} / (1- \gamma)$)
- utility is "**long term**" total reward

A fixed sequence of actions cannot be used in stochastic environments

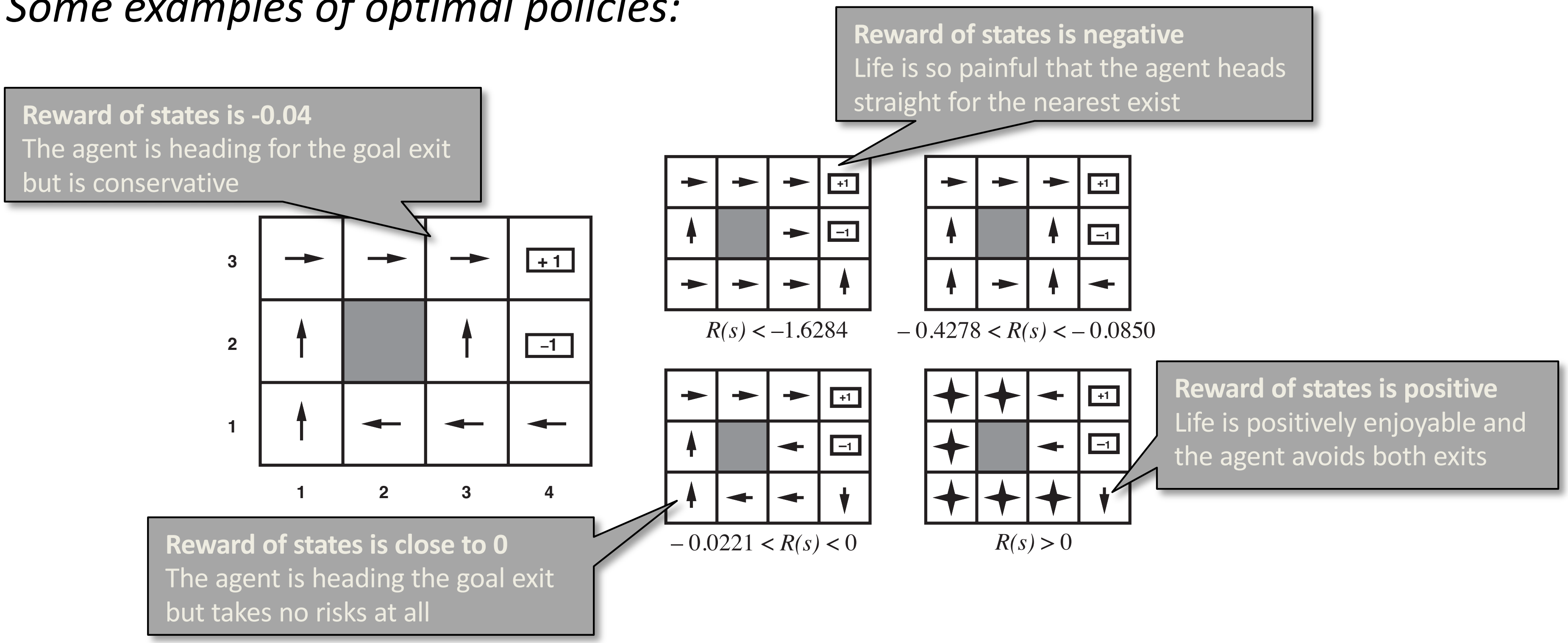
- agent might end up in a state other than the goal

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 – $\pi(s)$

- an **optimal policy** is a policy that yields the highest expected utility

Some examples of 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 $U(s) = U^{\pi^*}(s)$

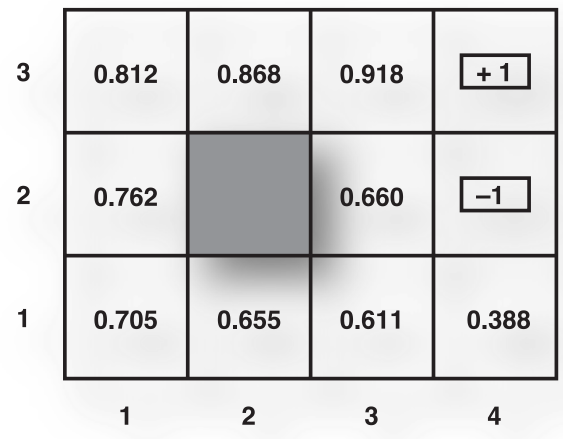
- then choose the action that maximizes the expected utility of the subsequent state

$\pi^*(s) = \text{argmax}_a \Sigma_{s'} P(s'|s,a)\, U(s')$

There is a direct relationship between the utility of a state and the utility of its neighbors:

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

This is called the **Bellman equation**.



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

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

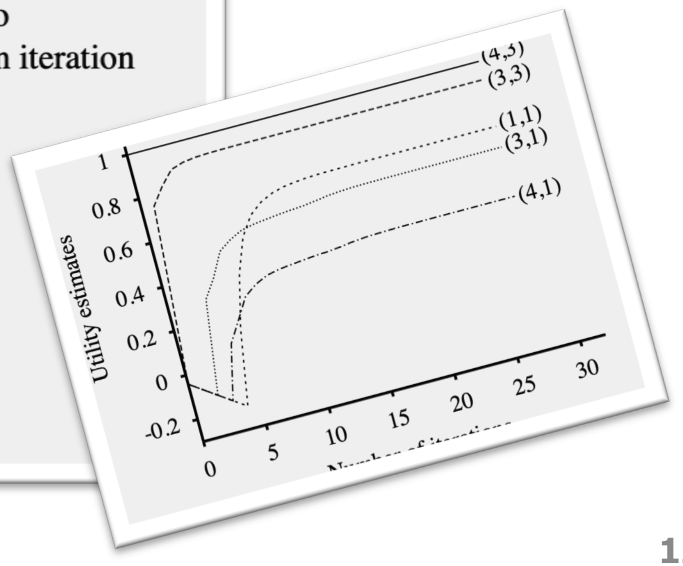
We can apply an **iterative approach**

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**)

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \Sigma_{s'} P(s'|s,a)\, U_i(s')$$

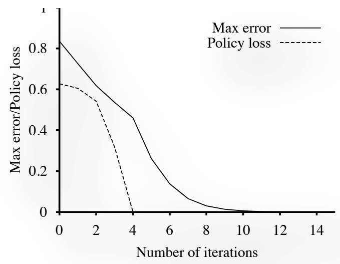
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
        rewards $R(s)$, discount γ
      ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
          δ, 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**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U



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

- policy loss measures the quality distance between the optimal utility and policy utility



We can iteratively improve the policy until no improvement is obtained.

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

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

policy evaluation – given a policy π_i, calculate U^{π_i}

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

- for small state spaces, solve Bellman equations exactly in $O(n^3)$
- for large state spaces, use value iteration

policy improvement – given a utility U^{π_i} calulate a new MEU policy π_{i+1}

$\pi_{i+1}(s) \leftarrow \operatorname{argmax}_a\ \Sigma_{s'}\ P(s' \mid s,a)\ U^{\pi_i}(s')$

Let us go from fully observable to partially observable environments (this is the real world).

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

Partially observable MDP (POMDP) is like MDP:

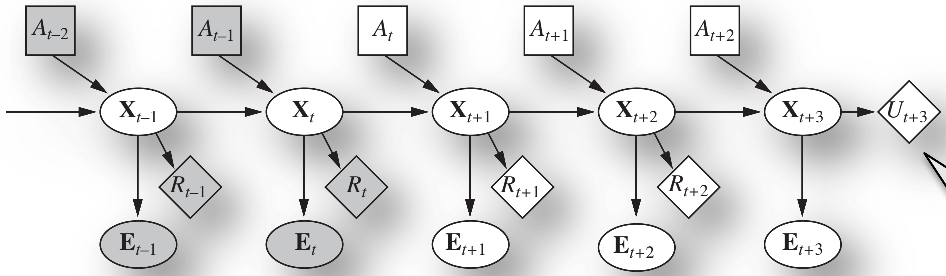
- the transition model P(s'|s,a)
- actions applicable in state A(s)
- reward function R(s)
- sensor model P(s|e)

We can use **belief states** instead of real states (recall that a believe state is probability distribution over all possible states).

Then, we can solve MDP over belief states - this requires modification of techniques to a continuous case.

The transition and sensors models are represented by a dynamic Bayesian network.

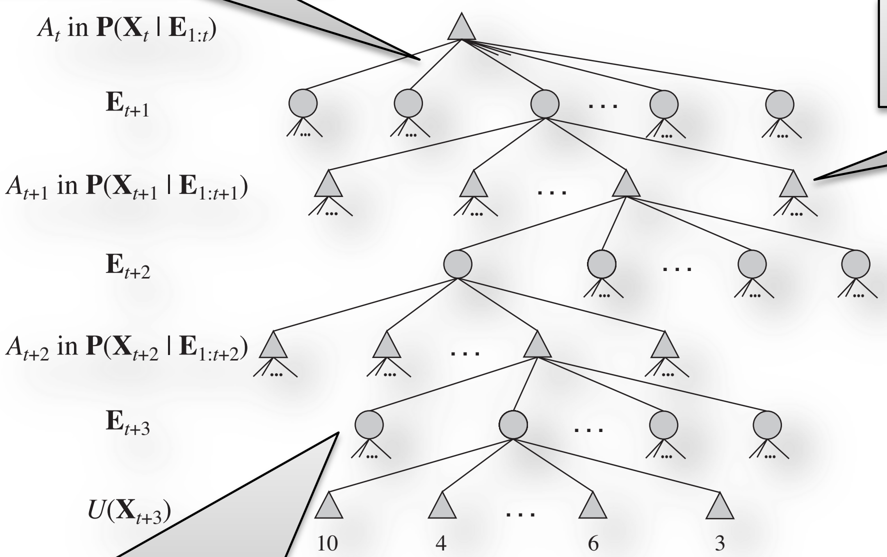
We add decision and utility nodes to get a **dynamic decision network**.



How far to enroll?
thanks to discount factor, the future rewards influence less the utility

Applicable actions

Believe state
(calculated using filtering, we know the path to it)



Chance nodes
(choices by the environment, what evidence arrives)

Decisions are done by projecting forward possible action sequences and choosing the best one.

This can be done by search using approach similar to **ExpectedMiniMax algorithm** for stochastic games.

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

Utility theory describes what agent wants.

Decision theory describes what agent should do.

Rational agent selects an action **maximizing expected utility**: action = $\text{argmax}_a \text{ EU}(a|\mathbf{e})$

Decision networks provide formalism for simple decisions.

Markov Decision Process is a formalism for sequential decision problems.

- solution of MDP is a **policy**
- optimal policies can be found by **value iteration** and **policy iteration** algorithms

Partially observable MDP extends MDP to partially observable environments.

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