Fundamentals of Artificial Intelligence – Rational Decisions Over Time

Matthias Althoff

TU München

Winter semester 2022/23

Organization

1. Markov Decision Processes
2. Value Iteration
3. Policy Iteration

The content is covered in the AI book by the section "Making Complex Decisions".

Learning Outcomes

- You understand the concept of *Markov decision processes* (MDP) and *partially observable Markov decision processes* (POMDP) and can assess which concept to use for a given problem.
- You understand the concept of an *optimal policy* and how it differs from an *optimal sequence*.
- You can create utility functions of state sequences for finite and infinite time horizons.
- You know and understand the *Bellman Principle of Optimality*.
- You can create and evaluate *Bellman equations* for a given stochastic optimization problem.
- You can solve Bellman equations using *value iteration* and *policy iteration*.

Overview of Probabilistic Methods

This lecture focuses on actions in dynamic environments.

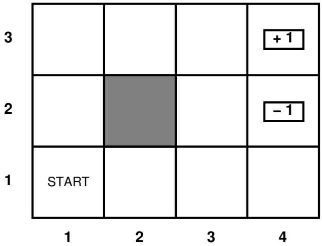
	Static environment	**Dynamic environment**
Without actions	Bayesian networks (lecture 9)	Hidden Markov models (lecture 10)
With actions	Decision networks (lecture 11)	Markov decision processes (lecture 12)

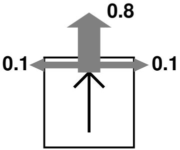
Variations of Markov Models

- Last lecture was on decision making of episodic environments.
- This lecture is about decision making in sequential environments, requiring us to consider dynamics.
- Previous models of sequential environments did not consider the possibility of taking actions:

		Control over state transitions?	
		no	**yes**
States fully obser-vable?	**yes**	Markov chain	Markov Decision Process (MDP)
	no	Hidden Markov model (HMM)	Partially Observable Markov Decision Process (POMDP)

Example of a Markov Decision Process (MarkovDecisionProcess.ipynb)





- States $s \in S$, actions $a \in A = \{Up, Down, Left, Right\}$.
- **Model** $P(s'|s,a) =$ probability that a in s leads to s'.
- **Reward function (or utility function)** $R(s,a,s')$ (or $R(s)$, $R(s,a)$)
$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$
- We assume the environment is fully observable: the agent always knows where it is.

Markov Decision Process (MDP)

A **Markov decision process** is a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and rewards.

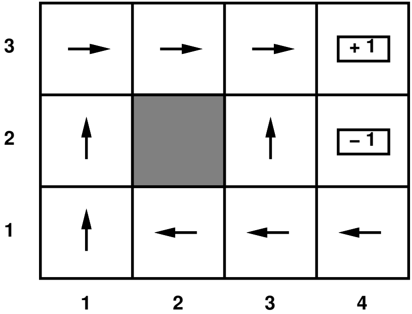
Formal definition

A Markov decision process is a 5-tuple (S,A,P,R,γ), where

- S is a finite set of states,
- A is a finite set of actions (alternatively, $A(s)$ is the finite set of actions available from state s),
- $P_a(s,s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t+1$,
- $R(s,a,s')$ is the immediate reward (or expected immediate reward) received after transition to state s' from state s with action a,
- $\gamma \in [0,1]$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

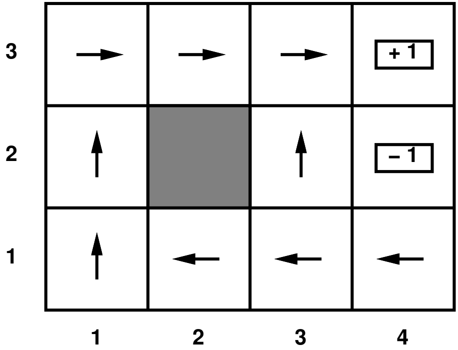
Solving MDPs

- In search problems, the aim is finding an optimal *sequence*.
- In MDPs, the aim is finding an optimal *policy* $\pi(s)$, i.e., the best action for every possible state s (because one can't predict where one will end up).
- The optimal policy maximizes the *expected utility*.
- Optimal policy when state penalty is $R(s, a, s') = R(s) = -0.04$:



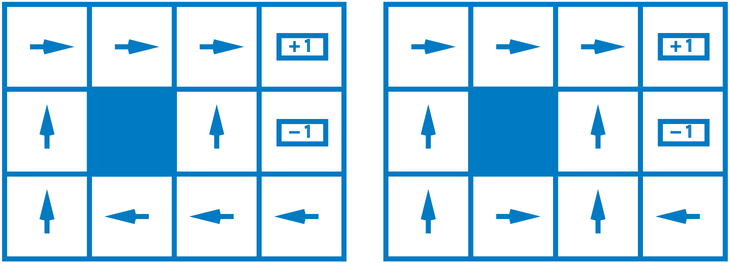
Tweedback Question

Does the optimal policy change when $R(s)$ is changed?



Tweedback Question

What is the missing reward value?

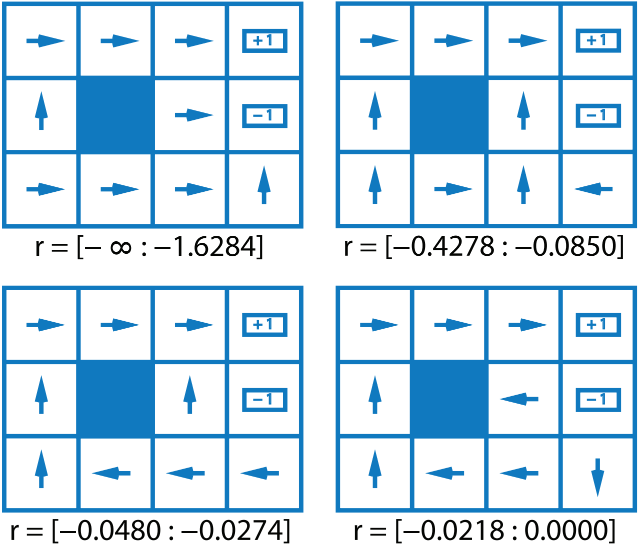


$r = -0.04$ $\quad$ $r = ?$

A -0.09,

B -0.02.

Optimal Policy for Different Ranges of $R(s)$



Motivating Examples

Not relevant for the exam

Some examples where MDPs have been applied:

- Agriculture
- Water resources
- Inspection, maintenance, and repair
- Purchasing, inventory, and production
- Finance and investment
- Queues
- Sales promotion
- Motor insurance claims
- Overbooking
- Epidemics
- Sports
- Patient admissions
- Design of experiments
- etc.

Utility of State Sequences

- We need to understand preferences between *sequences* of states.
- Typically consider **stationary preferences** on reward sequences:

$$[r, r_0, r_1, r_2, \ldots] \succ [r, r'_0, r'_1, r'_2, \ldots] \Leftrightarrow [r_0, r_1, r_2, \ldots] \succ [r'_0, r'_1, r'_2, \ldots]$$

Utility of sequences

There are only two coherent ways to combine rewards over time:

1. *Additive* utility function:
   $U([s_0, a_0, s_1, a_1, s_2, \ldots]) =$
   $R(s_0, a_0, s_1) + R(s_1, a_1, s_2) + R(s_2, a_2, s_3) + \cdots$
2. *Discounted* utility function:
   $U([s_0, a_0, s_1, a_1, s_2, \ldots]) =$
   $R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + \cdots$
   where γ is the **discount factor**

Utility of States

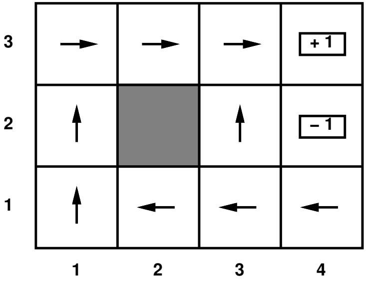
Utility of a *state* is defined to be

> $U(s) =$ expected (discounted) sum of rewards (until termination) assuming optimal actions.

Optimal policy of a state s and action space $A(s)$:

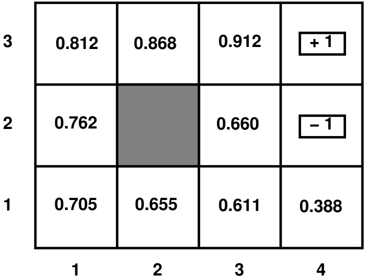
$$\pi^*(s) = \underset{a \in A(s)}{\texttt{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')].$$

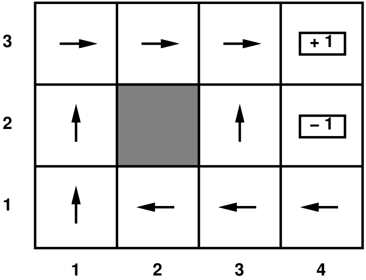
	1	2	3	4
3	0.812	0.868	0.912	+ 1
2	0.762		0.660	– 1
1	0.705	0.655	0.611	0.388



Optimal Policy for one State: Example (MarkovDecisionProcess.ipynb)

$$\pi^*(s) = \underset{a \in A(s)}{\texttt{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')].$$





Square (3,1), $\gamma = 1$:

$$\begin{aligned}
&\text{up:} && 0.8\,[-0.04 + 0.660] + 0.1\,[-0.04 + 0.655] + 0.1\,[-0.04 + 0.388] \approx 0.672\\
&\text{left:} && 0.8\,[-0.04 + 0.655] + 0.1\,[-0.04 + 0.660] + 0.1\,[-0.04 + 0.611] \approx 0.691\\
&\text{down:} && 0.8\,[-0.04 + 0.611] + 0.1\,[-0.04 + 0.655] + 0.1\,[-0.04 + 0.388] \approx 0.633\\
&\text{right:} && 0.8\,[-0.04 + 0.388] + 0.1\,[-0.04 + 0.660] + 0.1\,[-0.04 + 0.611] \approx 0.478
\end{aligned}$$

Additive Utility For Infinite Horizons

Problem: Infinite lifetimes $\Rightarrow$ additive utilities are infinite.

1. **Finite horizon**: termination at a *fixed time* T
   $\Rightarrow$ **nonstationary** policy: $\pi(s)$ also depends on time left.
2. **Absorbing state(s)**: with probability 1, agent eventually "dies" for any π
   $\Rightarrow$ expected utility of every state is finite.
3. **Discounting**: assuming $\gamma < 1$, $R(s) \leq R_{\max}$,

   $$U([s_0, a_0, s_1, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq R_{\max}/(1-\gamma) \quad \text{(geometric series)}$$

   Smaller $\gamma \Rightarrow$ shorter horizon.
4. **Maximize system gain**: System gain = Average reward per time step
   Infinite sequences can be compared in terms of **average reward**.
   The analysis of average-reward algorithms is beyond the scope of this lecture.

In sum, discounting is often the most useful technique.

Bellman Principle of Optimality

Bellman Principle of Optimality (Bellman, 1957)

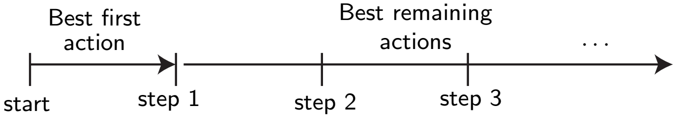
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Example:

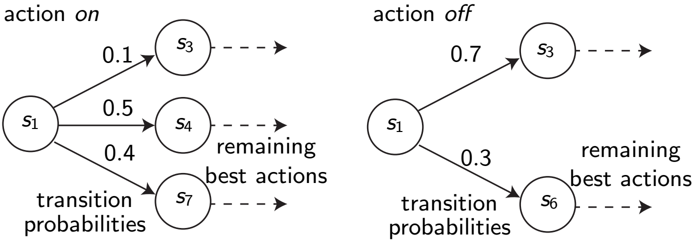
- To optimally get from Garching Forschungszentrum to Olympiazentrum with the subway for a party at Olydisco, you have to pass Studentenstadt.
- The optimal path from Studentenstadt to Olympiazentrum must be a partial optimal path from Garching Forschungszentrum to Olympiazentrum.

Bellman Principle of Optimality: Example

Basic idea:



Example: two actions *on* and *off*, $R(s, a, s') = R(s)$, $\gamma = 1$



Utility of state is utility when applying best actions starting in that state:

$$U(s_1) = \max\Big(0.1\,[R(s_1) + U(s_3)] + 0.5\,[R(s_1) + U(s_4)] + 0.4\,[R(s_1) + U(s_7)],$$

$$0.7\,[R(s_1) + U(s_3)] + 0.3\,[R(s_1) + U(s_6)]\Big)$$

Bellman Equation

Definition of utility of states leads to a simple relationship among utilities of neighboring states based on the *Bellman Principle of Optimality*:

Bellman equation (1957)

expected sum of rewards = current reward + $\gamma \times$ expected sum of rewards after taking best action:

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]. \tag{1}$$

Example: Bellman equation of the 4×3 world for state $(1,1)$, $\gamma = 1$:

$$\begin{aligned} U(1{,}1) = \max\big(& 0.8\,[-0.04 + U(1,2)] + 0.1\,[-0.04 + U(2,1)] + 0.1\,[-0.04 + U(1,1)],\ (\mathit{up}) \\ & 0.9\,[-0.04 + U(1,1)] + 0.1\,[-0.04 + U(1,2)],\ (\mathit{left}) \\ & 0.9\,[-0.04 + U(1,1)] + 0.1\,[-0.04 + U(2,1)],\ (\mathit{down}) \\ & 0.8\,[-0.04 + U(2,1)] + 0.1\,[-0.04 + U(1,2)] + 0.1\,[-0.04 + U(1,1)]\big)\,(\mathit{right}) \end{aligned}$$

Using the values from slide 14, one obtains that *up* is the best action.

Bellman Equation: Alternative Derivation

An alternative derivation is to first consider finite sequences. Let us compute the utilities of an agent living for

- **one time step:**

$$U_1(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a) R(s,a,s');$$

- **two time steps:**

$$U_2(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_1(s')];$$

- *i* **time steps (Bellman update):**

$$U_i(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_{i-1}(s')].$$

For $i \to \infty$ the above formula converges to the Bellman equation, which is also referred to as the value iteration algorithm presented next.

Q-Function

The Q-function is a useful generalization of the utility to simplify notation.

Repetition: utility

The expected utility when starting in a given state.

Q-function (aka action-utility function)

The expected utility when starting in a given state with a given action.

Optimal policy

→	→	→	+1
↑		↑	−1
↑	←	↑	←

Utility ($\gamma = 0.95$)

0.61	0.72	0.84	+1
0.50		0.41	−1
0.40	0.29	0.25	−0.07

Q-function ($\gamma = 0.95$)

.52 .52 .61 .47	.63 .55 .72 .62	.71 .61 .84 .49	+1
.50 .42 .43 .35		.41 .22 −.78 −.04	−1
.40 .34 .27 .32	.06 .29 .11 −.02	.25 −.06 −.28 .07	−.81 −.07 −.51 −.56

$(0.x \hat{=} .x)$

Bellman Equation of the Q-Function

From the definition of the Q-function, the utility is easily obtained:

$$U(s) = \max_{a \in A(s)} Q(s,a). \qquad (2)$$

The optimal policy can be directly obtained from the Q-function:

$$\pi^*(s) = \underset{a \in A(s)}{\mathtt{argmax}} Q(s,a).$$

Bellman equation for Q-functions

Comparing (1) with (2) results in

$$\begin{aligned} Q(s,a) &= \mathtt{Q-Value}(mdp, s, a, U) \\ &\overset{(1),(2)}{=} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')] \\ &\overset{(2)}{=} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in A(s')} Q(s',a')]. \end{aligned}$$

Value Iteration Algorithm (MarkovDecisionProcess.ipynb)

Problem: Solving the Bellman equation requires solving n **nonlinear** equations in n unknowns.

Idea:

- Start with arbitrary utility values (except for terminal states)
- Update to make them **locally consistent** with Bellman equation.
- Everywhere locally consistent $\Rightarrow$ global optimality.

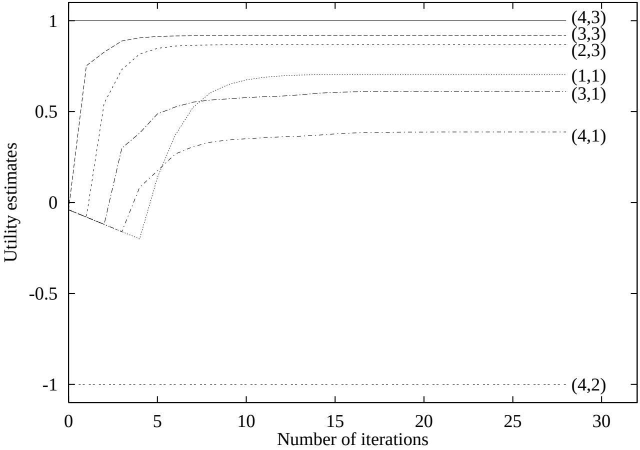
Algorithm:

- terminal state: $U(s) = R(s)$
- other states: Repeat for every s simultaneously until "no change"

$$U(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')] \qquad \text{for all } s$$

Value Iteration Algorithm: Example (MarkovDecisionProcess.ipynb)

Results for 4×3 world:



Convergence of Value Iteration

Max norm (aka infinity norm $\|U\|_\infty$)

We define the max-norm as $\|U\| = \max_s |U(s)|$

Consequently, $\|U - V\|$ = maximum difference between U and V.

Recall: Let U_i and U_{i+1} be successive approximations to the true utility U.

Convergence/Contraction

The approximation U_i converges to the true value U (proof on next slide)

$$\|U_{i+1} - U\| \leq \gamma \|U_i - U\|$$

Maximum Error

If $\|U_{i+1} - U_i\| < \epsilon$, then $\|U_{i+1} - U\| < 2\epsilon\gamma/(1-\gamma)$
I.e., once the change in U_i becomes small, we are almost done.

Proof of Convergence

Not relevant for the exam

Theorem: $\|U_{i+1} - U\| \leq \gamma \|U_i - U\|$

Proof: $\|U_{i+1} - U\|$

$$\overset{Def.}{=} \gamma \max_s \left| \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') - \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s') \right|$$

$$\overset{C1}{\leq} \gamma \max_s \max_{a \in A(s)} \left| \sum_{s'} P(s'|s,a) U_i(s') - \sum_{s'} P(s'|s,a) U(s') \right|$$

$$= \gamma \max_s \max_{a \in A(s)} \sum_{s'} P(s'|s,a) \left| U_i(s') - U(s') \right|$$

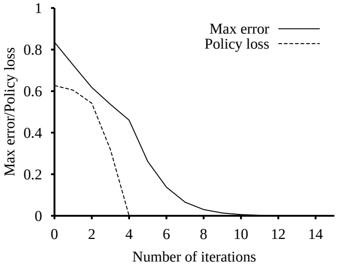
$$\overset{C2}{\leq} \gamma \max_s |U_i(s) - U(s)| = \gamma \|U_i(s) - U(s)\|$$

- Comment C1: $|\max f(x) - \max g(x)| \leq \max |f(x) - g(x)|$.
- Comment C2: $P(s'|s,a)$ are non-negative and sum to one.

Policy Iteration: Basic Idea (MarkovDecisionProcess.ipynb)

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

Right figure: Maximum error $\|U_i - U\|$ plotted with policy loss $\|U^{\pi_i} - U\|$ of the 4×3 world.



Policy iteration

- **Policy evaluation**: Given a policy π_i, calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.
- **Policy improvement**: Calculate a new policy π_{i+1} using a one-step look-ahead based on U_i using
  $\pi_{i+1}(s) = \underset{a \in A(s)}{\mathtt{argmax}} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$.

Terminates when the policy improvement yields no better policy.

Policy-Iteration Algorithm (MarkovDecisionProcess.ipynb)

function Policy-Iteration (mdp) **returns** a policy

inputs: mdp, a Markov decision process with states S, actions $A(s)$, transition model $P(s'|s,a)$

local variables: U, a vector of utilities for states in S, initially 0
π, a policy vector indexed by state, initially random

repeat
  $U \leftarrow \texttt{Policy}-\texttt{Evaluation}(\pi, U, mdp)$
  $unchanged \leftarrow true$
  for each state s **in** S **do**
    $a^* \leftarrow \arg\max_{a \in A(s)} \texttt{Q}-\texttt{Value}(mdp, s, a, U)$ **if**
    $\texttt{Q}-\texttt{Value}(mdp, s, a^*, U) > \texttt{Q}-\texttt{Value}(mdp, s, \pi(s), U)$ **then**
      $\pi(s) \leftarrow a^*$
      $unchanged \leftarrow false$
until $unchanged$
return π

Policy Evaluation

- The policy improvement step is straightforward.
- How do we implement the `Policy-Evaluation` routine?
- It is actually easier than solving the standard Bellman equations, since the action in each state is fixed by the policy:

$$U_i(s) = \sum_{s'} P(s'|s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')].$$

- For instance, for the policy on slide 8 we have $\pi(1,1) = up$, $\pi(1,2) = up$, and so on, so that the simplified Bellman equations are

$$\begin{aligned} U_i(1,1) =& 0.8\,[-0.04 + U_i(1,2)] + 0.1\,[-0.04 + U_i(2,1)] \\ &+ 0.1\,[-0.04 + U_i(1,1)], \\ U_i(1,2) =& 0.8\,[-0.04 + U_i(1,3)] + 0.2\,[-0.04 + U_i(1,2)], \\ & \dots \end{aligned}$$

  These equations are linear! (since max() operator is removed)
- The linear equations can be solved in $\mathcal{O}(n^3)$ time (n: nr. of states).

Decision Making in Partially Observable Environments

- A Markov decision process (MDP) can be viewed as

$$\boxed{\text{MDP} = \text{Markov chain} + \text{actions} + \text{rewards}.}$$

- In reality, the agent does not know exactly in what state it is when the environment is **partially observable**.
- Consequently, the agent cannot execute a policy of the form $\pi(s)$, since s is not exactly known.
- After introducing a sensor model, we obtain **partially observable Markov decision processes** (POMDPs), which can be viewed as

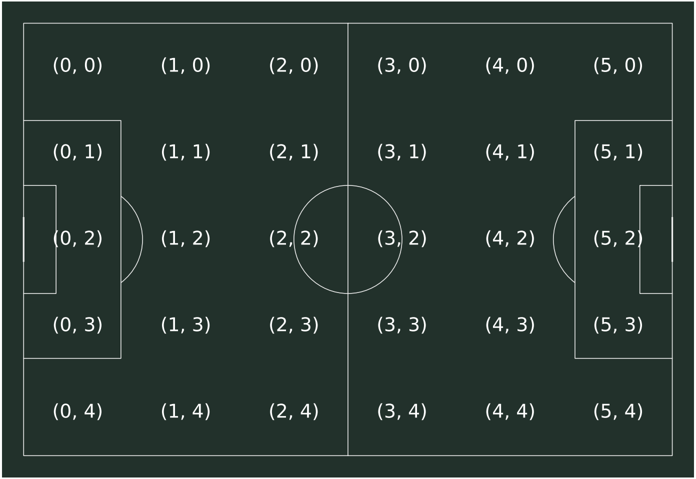
$$\boxed{\text{POMDP} = \text{hidden Markov model} + \text{actions} + \text{rewards}.}$$

Details on POMDPs can be found in the bonus lecture:

- Not relevant for Techniques in Artificial Intelligence (IN2062);
- Possibly relevant for Fundamentals of Artificial Intelligence (IN2406) (will be decided later).

Real-World Example: Soccer Modelling

We consider a 6x5 grid discretization of a soccer pitch:



Assumptions (1/2)

- At each discrete timestep, the attacking side (i.e., the agent) can either shoot or attempt to pass the ball to any other cell.
- Success probabilities for shooting (i.e., scoring) and passing (i.e., transitioning to the target cell without losing possession) are based on data from the 2019/20 season of the English Women's Super League[1].
- State space $S = \{(0,0),(0,1),...,\mathit{Goal},\mathit{PossessionLost}\}$, with *Goal* and *PossessionLost* being terminal states.
- Action space $A = \{(0,0),(0,1),...,\mathit{Shoot}\}$.
- For simplicity, we disregard the probability of a player passing (or shooting) to a different cell than intended:

$$P(s' = (x,y)|a = (x,y), s = s_0) = 1 - P(s' = \mathit{PossessionLost}|a = (x,y), s = s_0)$$
$$P(s' = \mathit{Goal}|a = \mathit{Shoot}, s = s_0) = 1 - P(s' = \mathit{PossessionLost}|a = \mathit{Shoot}, s = s_0)$$

[1] `https://github.com/statsbomb/open-data`

Assumptions (2/2)

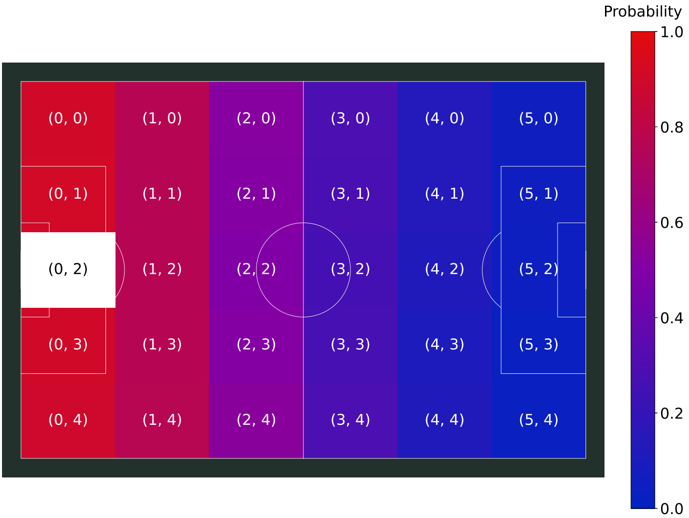
- Discount factor $\gamma = 0.99$.
- Reward function $R(s')$
$$= \begin{cases} 10 & \text{if } s' = \mathit{Goal} \\ -0.002 & \text{if } s' = \mathit{PossessionLost} \\ -0.001 & \text{otherwise (living penalty to incentivize progress)} \end{cases}$$

Expected Goals $P(s' = Goal | a = Shoot, s = s_0)$

Probability of scoring when shooting from a given cell s_0:

Transition Probabilities $P(s' = (x, y) | a = (x, y), s = s_0)$

For $s_0 = (0, 2)$:



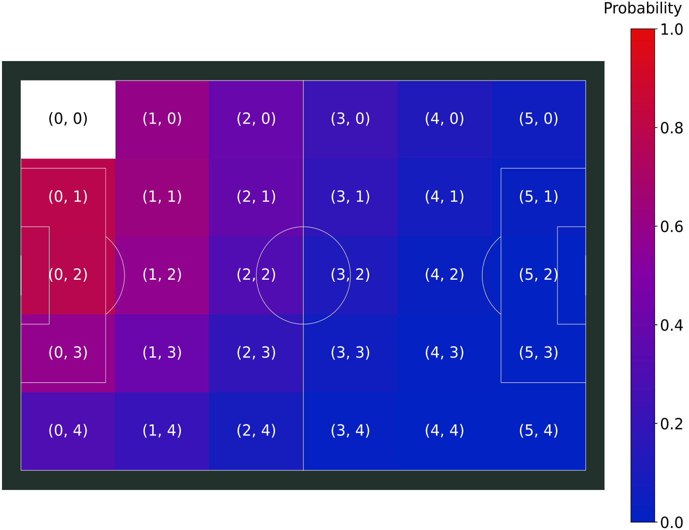
Transition Probabilities $P(s' = (x, y) | a = (x, y), s = s_0)$

For $s_0 = (2, 0)$:

Transition Probabilities $P(s' = (x, y) | a = (x, y), s = s_0)$

For $s_0 = (3, 3)$:

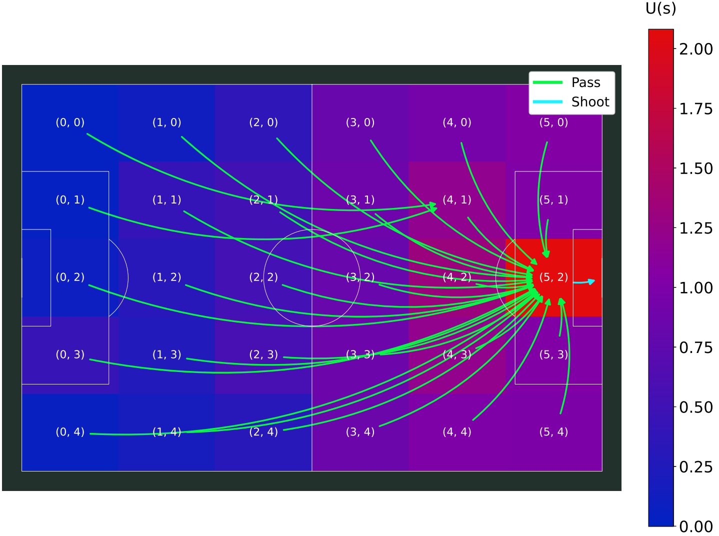
Transition Probabilities - Video Animation



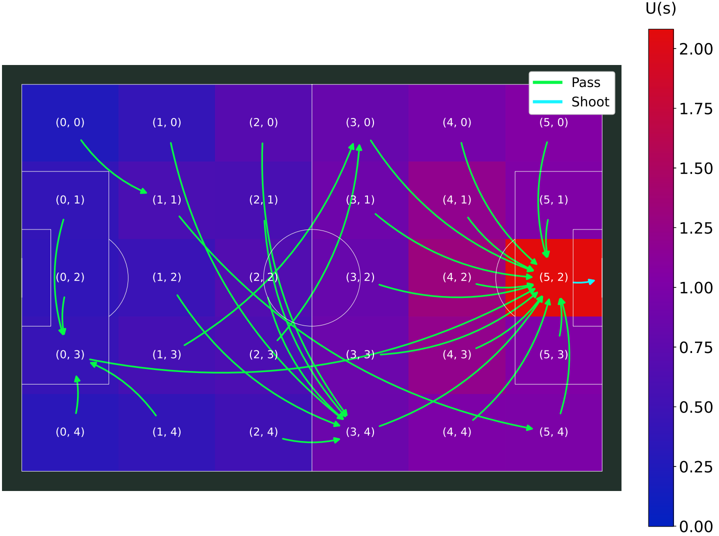
Solution Method

- We use the `Policy-Iteration` algorithm to compute the optimal strategy.
- The utility values are initialized to all zeros.
- The policy is initialized by randomly sampling from the action space.

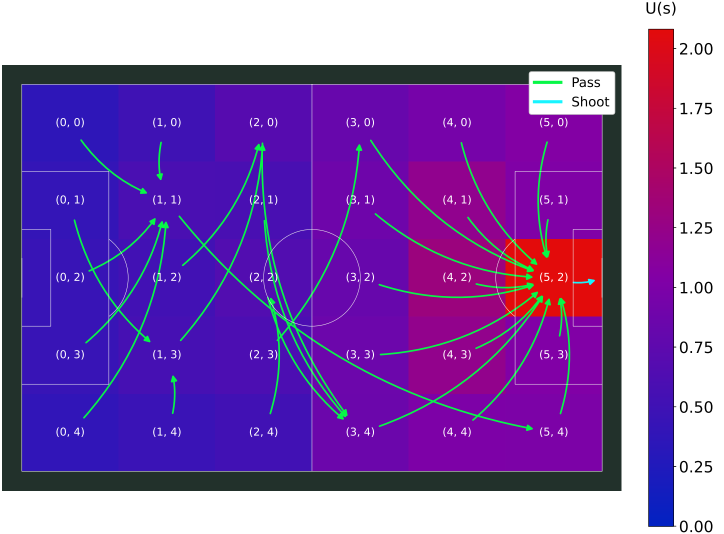
After 1 Iteration



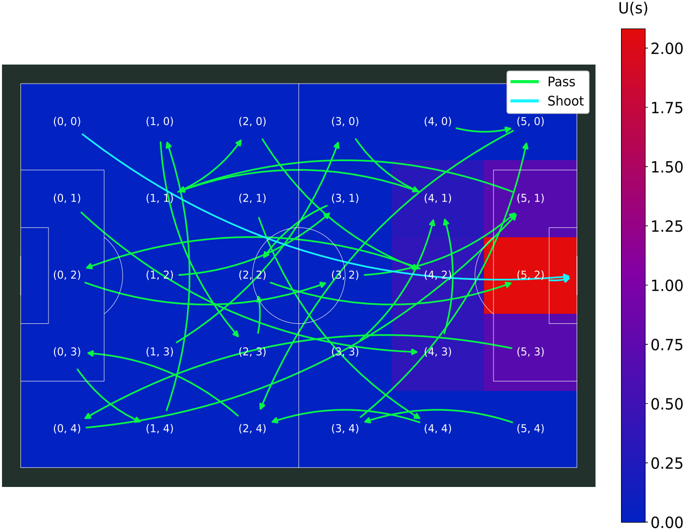
After 2 Iterations



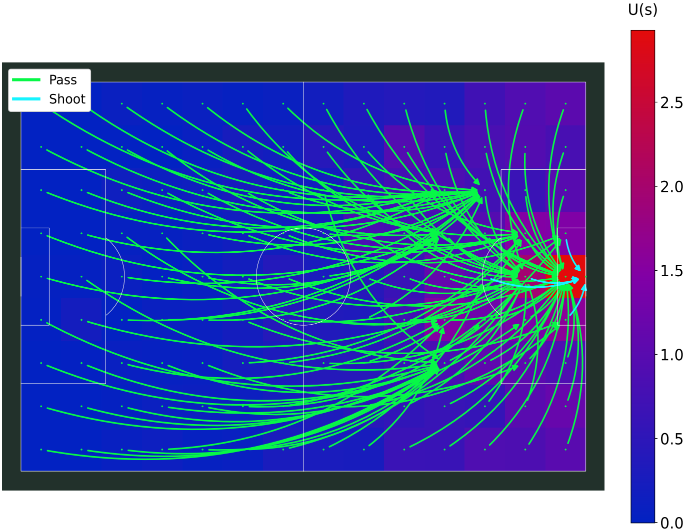
After Convergence



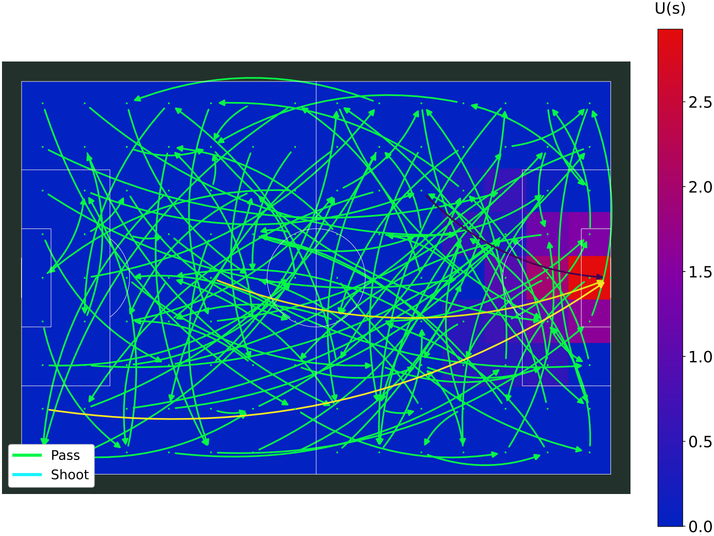
Policy Iteration - Video Animation



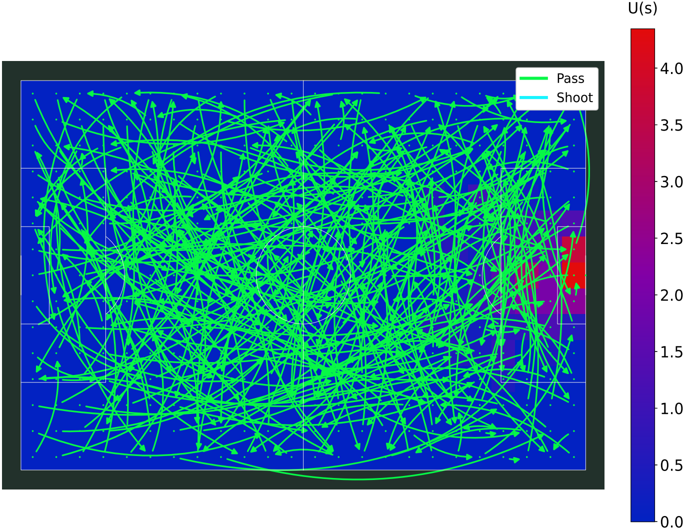
High Grid Resolution - Video Animation



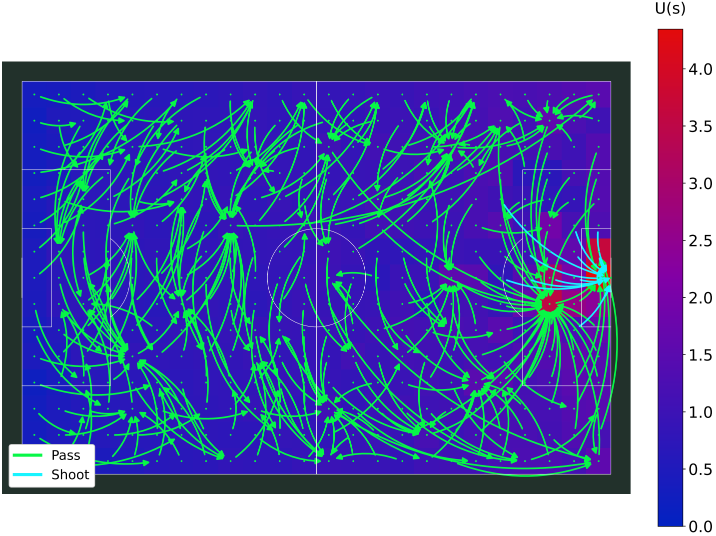
High Grid Resolution - After Convergence



Very High Resolution - Video Animation



Very High Resolution - After Convergence



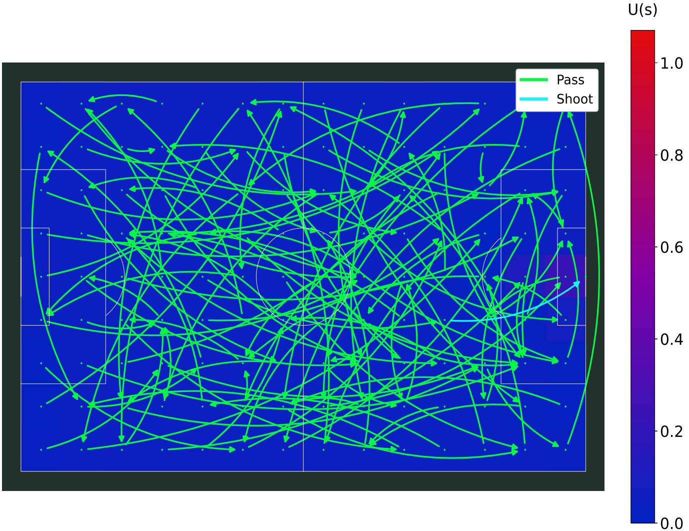
Experiment: Changing the Reward (1/2)

- Old reward function: $R(s')$
$$= \begin{cases} 10 & \text{if } s' = \mathit{Goal} \\ -0.002 & \text{if } s' = \mathit{PossessionLost} \\ -0.001 & \text{otherwise} \end{cases}$$

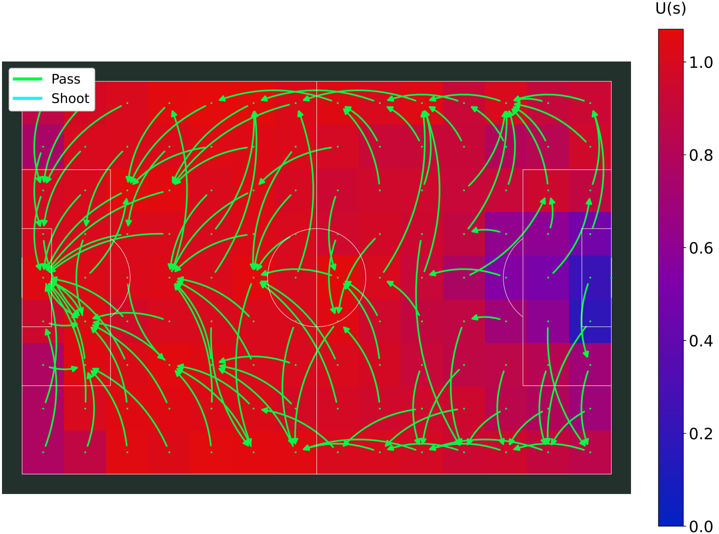
- New reward function: $R(s')$
$$= \begin{cases} 10 & \text{if } s' = \mathit{Goal} \\ -0.1 & \text{if } s' = \mathit{PossessionLost} \\ 0.05 & \text{otherwise} \end{cases}$$

- How will the optimal policy look like?

Optimal Policy - Video Animation (1/2)



Optimal Policy: Radically Possession-Oriented (1/2)



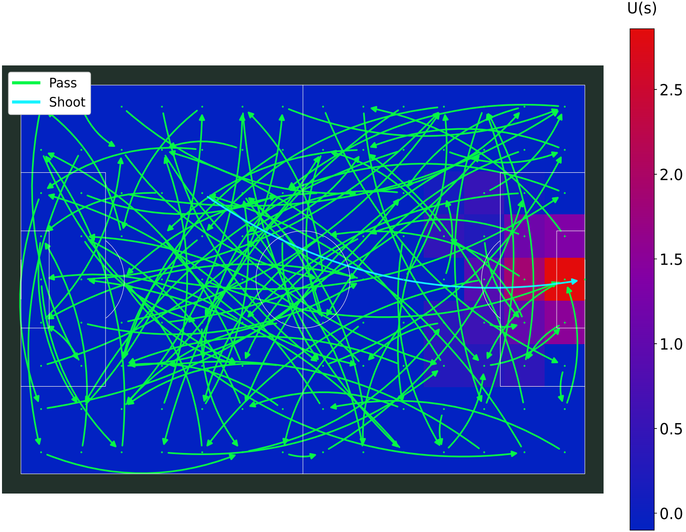
Experiment: Changing the Reward (2/2)

- Old reward function: $R(s')$
$$= \begin{cases} 10 & \text{if } s' = \mathit{Goal} \\ -0.002 & \text{if } s' = \mathit{PossessionLost} \\ -0.001 & \text{otherwise} \end{cases}$$

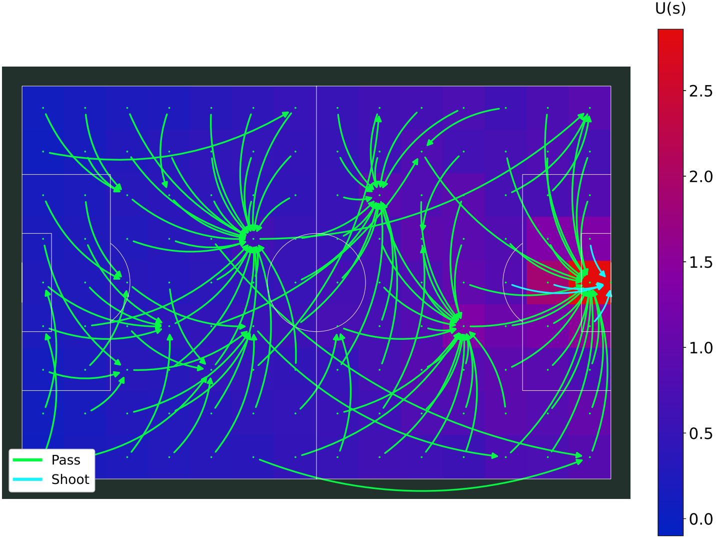
- New reward function: $R(s')$
$$= \begin{cases} 10 & \text{if } s' = \mathit{Goal} \\ -0.1 & \text{if } s' = \mathit{PossessionLost} \\ -0.1 & \text{otherwise} \end{cases}$$

- How will the optimal policy look like?

Optimal Policy - Video Animation (2/2)



Optimal Policy: Direct Play (2/2)

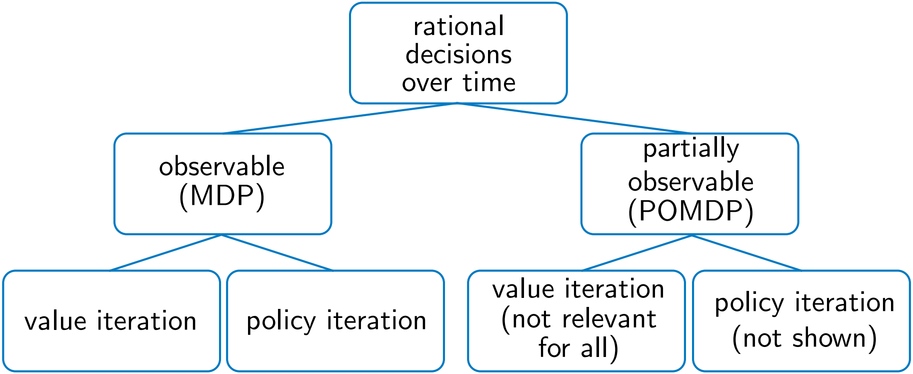


Soccer Modelling Example - Summary

- By formulating soccer as an MDP, we computed the optimal on-the-ball strategy using `Policy-Iteration`.
- The optimal play depends on the reward configuration (as well as other factors, such as γ).
- Based on the computed U and π, we can also derive performance metrics for both teams and individual players[2], with downstream usecases such as player scouting or betting.
- Further work could be
  - to also consider the opponent's actions on the ball by modeling the process as a *Markov game*[3];
  - to consider a continuous state space instead of the grid discretization (will be covered in the next lecture).

[2] **Van Roy et.al. (2020) - Valuing On-the-Ball Actions in Soccer**.
[3] **Hu and Wellman (1998) - Conjectural Equilibrium in Multiagent Learning**.

Overview of Making Rational Decisions Over Time



Summary

- Sequential decision problems in uncertain discrete environments can be modeled as Markov decision processes.
- The utility of a state sequence is the sum of all the rewards over the sequence, possibly discounted over time.
- The optimal solution of an MDP is a **policy** that associates a decision with every state that the agent might reach. A solution can be obtained by **value iteration**.
- **Policy iteration** converges faster since a policy might already be optimal without knowing the exact utilities of each state.