Lecture 2: Formal Models of AI Problems and Search

Viliam Lisý & **Branislav Bošanský**

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

bosansky@fel.cvut.cz

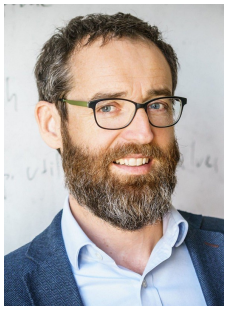
February, 2023

Course organisation

https://cw.fel.cvut.cz/wiki/courses/zui/start

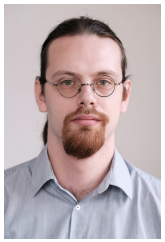
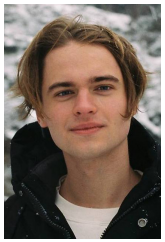
13 lectures leading towards some of the mentioned milestones





13 labs going deeper to selected algorithmic / theoretic topics




3 programming homeworks in python evaluated by BRUTE

Midterm test

Final written exam

Course grading

https://cw.fel.cvut.cz/b212/courses/zui/start

30% for programming homework

- 10% State space search (A*) algorithm
- 10% Reinforcement learning
- 10% Game playing bot
- Extra tasks for additional point possible

Each task must be submitted for $\geq 50\%$ of its points

Deadline penalties: $\leq$ 24h: -20%; $>$ 24h: 0

Plagiarism will not be tolerated!

If you have serious issues **let us know** ASAP.

15% for the midterm written test

55% for the final written exam

in case of 80+% overall, also a brief oral exam

Standard evaluation scale: https://fel.cvut.cz/en/education/rules/Study_and_Exam_Code.pdf

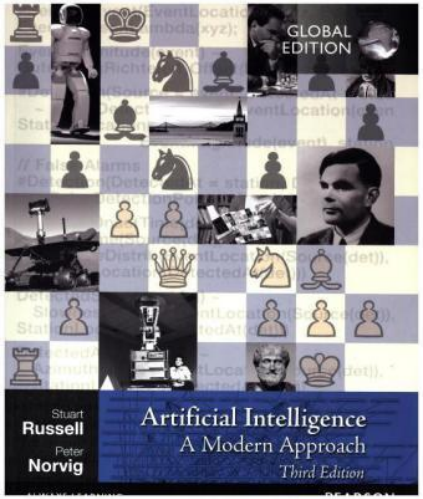
Course Topics Overview

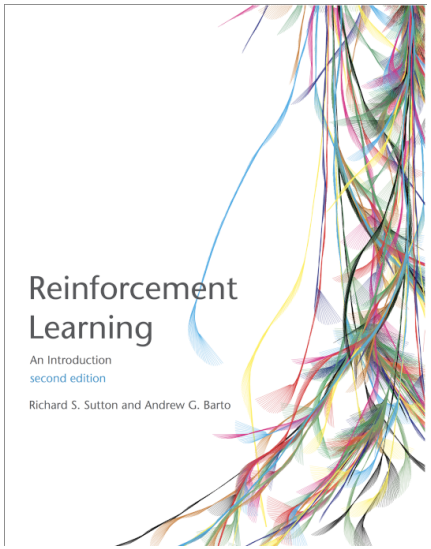
- Formal models of AI problems
- Search, A*
- Reinforcement Learning
- Two-Player Perfect-Information Games
- Logical Problem Representations
- Uncertainty in AI
- Sequential Decision Making with Limited Information

Course literature

Slides are not study materials!

1. Take notes.
2. Artificial Intelligence: A Modern Approach (AIMA) by Stuart J. Russell and Peter Norvig (however, it is not free)
3. Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto (PDF available online)
4. Links on the courseware page and in slides
5. Wikipedia

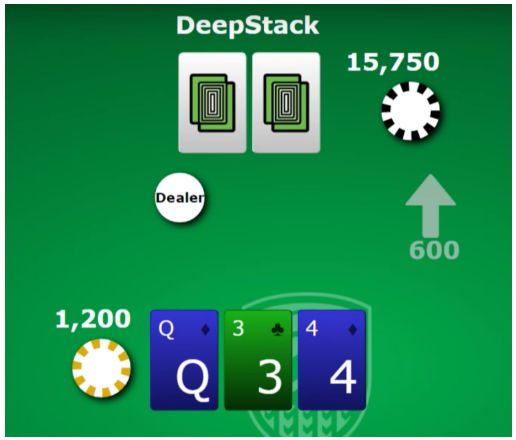




Key Points of the Lecture

- Why we are talking about search at all?
- Why it is good to have a formal representation of a problem?
- Provide a more unifying perspective on different algorithms.

Many (even recent) great AI breakthroughs use search as one of the components.



Search and AI

Search is one of the most fundamental and universal method for solving problems.

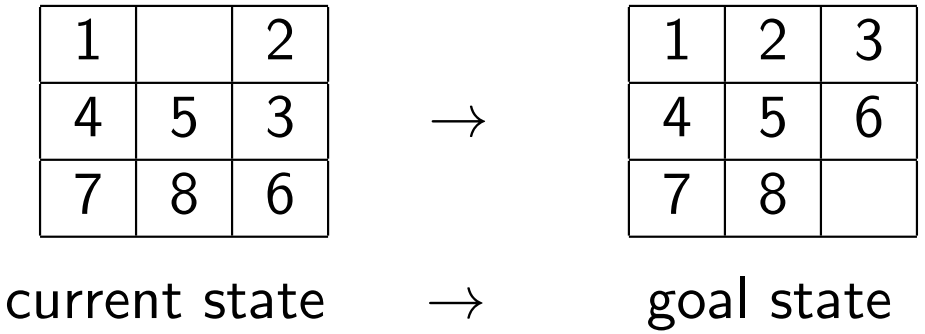
Having a **formal representation** of the problem, search algorithms allow us to look for a solution in a systematic way:

- formal representation of every possible situation in the scenario – the **states** of the problem (denoted S)
- how the states can be changed by the algorithm (agent) – the **actions** in the scenario (denoted A)

By applying an action $a \in A$ to a state $s \in S$, the state will change to a different state $s' \in S$.

States must contain all information necessary to determine applicable actions, transformation of the environment, or whether the goal has been reached.

Example 1 – 8 puzzle



Goal: rearrange the numbers by moving the empty square to adjacent squares so that the numbers are ordered

Possible representations:

- values of tiles in a sequence
  $s = [1, \_, 2, 4, 5, 3, 7, 8, 6]$
- position of numbers
  $s = [1, 3, 6, 4, 5, 9, 7, 8]$

Example 2 – Robotic arm

Find correct configurations of joints / parts of the arm so that the arm catches a desired object.

Possible representations:

- $s = [\theta_1, \alpha_1, \theta_2, \alpha_2, \ldots]$

Example 3 – Chess

Possible representations:

- positions of pieces on the board
  $s = [[A1, B1, C1, \ldots, H2], [A8, \ldots, H7], \ldots] \rightarrow$ additional information needed besides the board itself (king has moved, rook has moved, repeated positions (!))

Alternatively, a **history of played moves** represents a state.

Solution of a Problem

Many of the AI problems can be formulated as finding a sequence of actions that leads to a **goal state**.

We want to find the best such a sequence

- minimize the number of actions
- every action can have some cost (or reward) associated with it
  $\rightarrow$ minimization of total cost (maximization of total reward)

We can reason about possible states / effects of actions
(the rules of the environment are known (!)):

- we have a formal model
- (for the large scale) access to a simulator

Main AI Models for (Sequential) Decision Making

There are several fundamental models when searching for optimal sequence of actions based on searching through state space (possibly uncertain effect of actions / stochastic environment):

- Markov Decision Processes (MDPs)
- Partially Observable Markov Decision Processes (POMDPs)
- (Imperfect Information) Extensive-Form Games (EFGs)
- (Partially Observable) Stochastic Games (POSGs)

We introduce selected general models now to emphasize the importance of the correct formalization of the problem. Selected algorithms for solving them optimally will be introduced later.

Main AI Models for (Sequential) Decision Making

There are several fundamental models when searching for optimal sequence of actions based on searching through state space (possibly uncertain effect of actions / stochastic environment):

- Markov Decision Processes (MDPs) $\rightarrow$ perfectly observable environment, only 1 agent is acting
- Partially Observable Markov Decision Processes (POMDPs) $\rightarrow$ **partially observable environment**, only 1 agent is acting
- (Imperfect Information) Extensive-Form Games (EFGs) $\rightarrow$ perfectly (partially) observable environment, finite horizon, n **agents can act (every agent optimizes own goal / utility)**
- (Partially Observable) Stochastic Games (POSGs) $\rightarrow$ perfectly (partially) observable environment, **infinite horizon**, n agents can act (every agent optimizes own goal / utility)

Markov Decision Processes (MDPs)

Consider (finite) sets of states S, rewards R, and actions A. The agent interacts with the environment in discrete steps $t = 0, 1, 2, \ldots$. At each timestep, the agent receives the current state $S_t \in S$, selects an action based on the state $A_t \in A$. As a consequence of taking the action, the agent receives a reward $R_{t+1} \in R$ and find itself in a new state S_{t+1}.

Rewards and states are generated based on a *dynamics* of the MDP

$$p(s', r|s, a) \leftarrow Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

The next state depends only on the current state and the action (Markov property). **In the first lectures/labs, we assume that the environment is deterministic.**

Transition and reward dynamics can be defined separately.

Markov Decision Processes (MDPs) – Example

#	#	#	#	#	#
#	G				#
#	#	#	#		#
#	↓	#	#		#
#					#
#	#	#	#	#	#

Consider a robot (↓) in a maze (# are walls), the arrow represents the direction the robot is facing, G is gold.
What are the states and actions?

- $s = (X, Y, d, G)$
- actions = (move_forward, move_backward, turn_left, turn_right)

MDP dynamics:

- $p((1, 1, \downarrow, \text{false}), 0 \mid (1, 2, \downarrow, \text{false}), \text{move\_forward}) = 1$
- $p((1, 1, \downarrow, \text{false}), 0 \mid (1, 2, \downarrow, \text{false}), \text{move\_backward}) = 0$
- ...

Markov Decision Processes (MDPs)

Why do we need some generic formal description?

- we will have a well-defined problem (inputs / outputs for the algorithm)
- formalization helps to think about the problem (e.g., formalizing the dynamics)
- we can reuse existing algorithms
- if we design and implement a brand-new algorithm for MDPs (POMDPs / EFGs / ...), we can solve (almost) all instances

Partially Observable Markov Decision Processes (POMDPs)

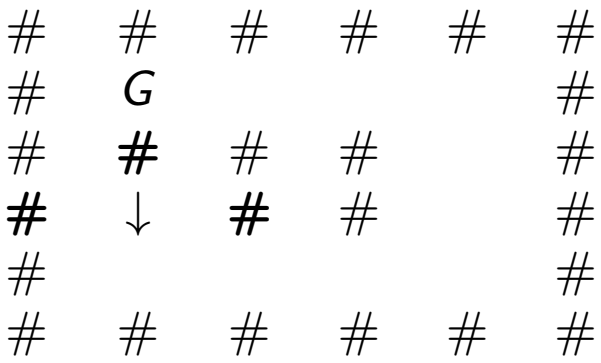
States, actions, and rewards are as before, however, the agent cannot perfectly observe the current state.

The agent has a **belief** – a probability distribution over states that express the (subjective) likelihood about the current state. The agent receives **observations** from a finite set O that affect the belief. The agent starts from an **initial belief** and based on actions and observations, it updates its belief. Given the current belief $b : S \to [0, 1]$ and some action $a \in A$ and received observation $o \in O$, the new belief is defined as:

$$b(s') = \mu O(o|s', a) \cdot \sum_{s \in S} Pr(s'|s, a) \cdot b(s)$$

where μ is a normalizing constant.

POMDP – Example



The robot can now perceive only its surroundings but does not know the exact position in the maze. States and actions remain the same.

- s $= (X, Y, d, G)$
- actions = (move_forward, move_backward, turn_left, turn_right)

Observations are all possible combinations of walls / free squares in the 4-neighborhood:

- $(\#, \#, \#, \#), (\#, \#, \#, \_), \ldots$

Extensive-Form Games (EFGs)

Agent is not the only one that changes the environment. Every state has a player that acts in that state.EFGs are typically visualized as game-trees that:

- are finite (the game has some pre-defined horizon; note that (PO)MDPs do not have this!)
- node of the game tree corresponds to the history of actions from the beginning, edges are actions (as search trees)
- rewards (termed utilities) are defined only in terminal states (leafs of the game tree)
- agent can have imperfect information (certain states can be indistinguishable) $\rightarrow$ we will not be able to cover this in ZUI ($\rightarrow$ B4M36MAS)

Partially Observable Stochastic Games (POSGs)

POSGs are a multi-agent extension of POMDPs $\rightarrow$ every agent can have their own actions, observations, and rewards. Every agent has its own belief (about the state, about beliefs of other agents, ...).

One of the most general formal model $\rightarrow$ algorithmically intractable in general.

Solution of a Deterministic MDP

How can we find a solution of a deterministic MDP?

Find the best sequence of actions leading to the goal $\rightarrow$ explore relevant states of an MDP and find the best action to be played in these states such that the trajectory (or a run)

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, \ldots, S_k$$

maximizes the accumulated reward (and S_k is a goal state)[1].

For now, the rewards are summed together (in case of stochastic transitions / POMDPs / negative costs, a discounted sum is typically used with discount factor $0 < \gamma < 1$).

[1] Maximization of rewards = minimization of costs (we will use both).

Solving Deterministic MDPs

1. Start from the initial state S_0
2. Apply available actions to the current state and generate new possible states
3. Select one of the newly generated states as the current one
4. If the current state is the goal state $\rightarrow$ finish
5. If not, go to step 2

Questions:

- Which state should we select out of all generated new states in step 3?
- What if we generate a state that we have already explored?

Solving Deterministic MDPs – Choosing Next State

Q1

Which state should we select out of all generated new states in step 3?

Goal is to find the best sequence of actions $\rightarrow$ we want to explore the ones with the highest rewards (lowest costs) first.

What if we make a mistake? $\rightarrow$ We cannot forget which other states are reachable $\rightarrow$ We keep a (sorted) list of reachable states that can be further explored – **open list** or **fringe**.

Solving Deterministic MDPs – Variants of Uninformed Search

Variants of using the **fringe**:

- the fringe is sorted, new states to explore are taken from the beginning $\rightarrow$ **uniform-cost search**
- the fringe is unsorted, newly expanded states are inserted to the front, new states to explore are taken from the beginning $\rightarrow$ **depth first search (DFS)**
- the fringe is unsorted, newly expanded states are appended at the back, new states to explore are taken from the beginning $\rightarrow$ **breadth first search (BFS)**

BFS is complete, finds the shallowest solution (the sequence that requires the least number of actions while ignoring rewards). Requires exponential memory (and time).

Solving Deterministic MDPs – Variants of Uninformed Search

DFS is not complete (the algorithm might not terminate) $\rightarrow$ we limit the maximal length of the sequence actions DFS can explore and iteratively increase this limit $\rightarrow$ **iterative deepening**.

Uniform-cost search is complete and optimal (in case all rewards are strictly negative). A variant of **Dijkstra's algorithm** (only the best path to a goal state not all states).

Solving Deterministic MDPs

Q2

What if we generate a state that we have already explored?

Using this algorithm, we are generating a **search tree**. Every node of the search tree corresponds to a state in the environment but multiple nodes can correspond to the same state.

We can maintain a **closed list** of already evaluated states.

Iterative Deepening

Combining good characteristics of BFS and DFS. Let's have a `limited-depth-dfs` method:

- call `limited-depth-dfs` with depth limit 0,
- if unsuccessful, call `limited-depth-dfs` with depth limit 1,
- if unsuccessful, call `limited-depth-dfs` with depth limit 2, etc.

Complete, finds the shallowest solution, space requirements of a DFS.Counterintuitively, it is not that wasteful (timewise):

- the search tree grows exponentially $\rightarrow$ it is more time consuming to generate / evaluate all states in depth exactly d than repeatedly visiting states in the shallower depth

What if we want to optimize cost instead of number of actions? $\rightarrow$ limit the overall cost and increase the cost iteratively by 1.

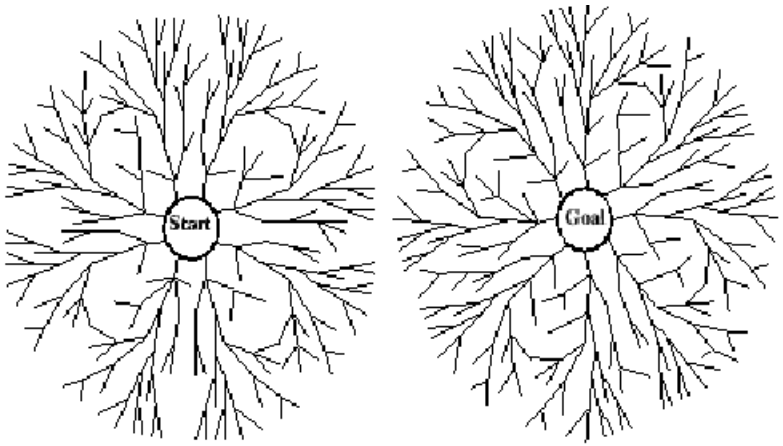
Backward / Bidirectional Search

Do we need to search only from the initial state? $\rightarrow$ No.

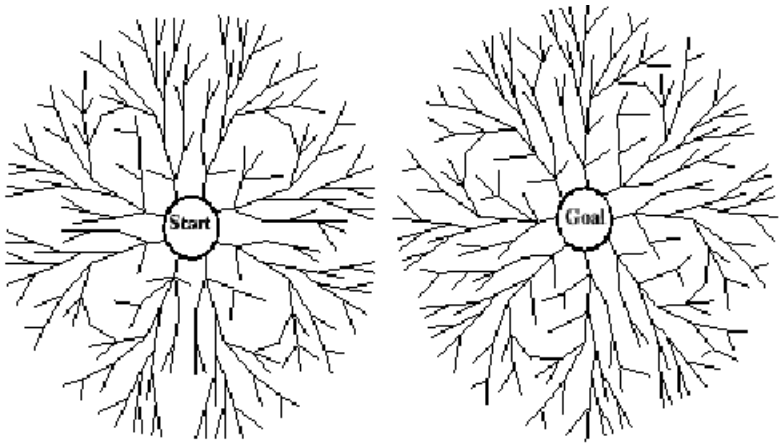
Sometimes, searching from the goal state to a starting state can be better:

- number of the actions that lead to the goal state is small (the problem is difficult at the beginning)
- we need to be able to effectively generate previous states

We can go even further $\rightarrow$ searching from the both sides.



Bidirectional Search



It is tempting $\rightarrow$ searching from start / goal (e.g., in parallel (!)).

If the shallowest solution has depth d, we can expand only $b^{d/2}$ nodes (where b is the branching factor (number of available actions)).

But what if the searches do not meet "in the middle"? $\rightarrow$ We'll see the next week.