Lecture 13: Sequential Decisions with Partial Information (POMDPs) 2

Viliam Lisý & **Branislav Bošanský**

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

viliam.lisy@fel.cvut.cz, **bosansky@fel.cvut.cz**

May, 2024

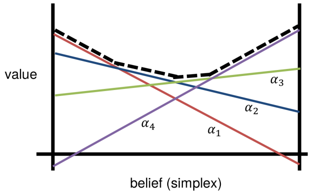
Reminder: Representation of Value Function with α vectors

If we fix an action $a \in \mathcal{A}$, the value function (the expected reward after playing that action) is a **linear function** in the current belief. These linear functions are called α-**vectors**.

For each belief point, we take the best action hence we maximize over all α-vectors:

$$v(b) = \max_{\alpha \in V} \sum_{s \in \mathcal{S}} \alpha(s) \cdot b(s)$$

where $\alpha(s)$ corresponds to the value of the linear function α in state s.



In general, α-vectors represent expected value for a **policy** (contingency plan consisting of multiple steps).

Exact value iteration in POMDPs

In exact (full) value iteration in POMDPs, $|V_t| = |\mathcal{A}| \cdot |V_{t-1}|^{|\mathcal{O}|}$ new α-vectors are generated in each step of the algorithm.

V_t converges to optimal value function (the algorithm incrementally constructs all possible t-step policies).

It is clear that such approach will not scale well. Pruning dominated α-vectors is possible but does not solve the issue.

Observation

We do not need to compute all α-vectors – large portion of belief space is (often) not reached hence not relevant for solving the problem.

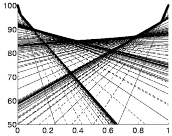
We can keep only a bounded number of belief points and for each belief point we keep 1 (the best) α-vector.

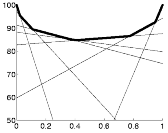
Point-based updates and point-based value iteration (PBVI)

Let $\mathcal{B} = \{b^1, b^2, \dots\}$ be a set of $|\mathcal{B}|$ belief points. **Point-based value iteration** performs Bellman update only for this limited set of belief points:

- instead of adding all α-vectors, only the α-vectors that are optimal in some of the belief points from $\mathcal{B}$ are kept,
- we perform standard update for a limited set of belief points $b \in \mathcal{B}$:

$$v_{t+1}(b) = \max_a \left\{ \sum_{o \in O} \max_{\alpha' \in v_t} \left[\sum_{r,s,s'} \mu p(s', r | s, a) b(s) O(o | s', a) \left(r + \gamma \alpha'(s') \right) \right] \right\}$$



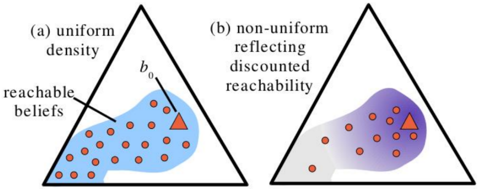


Comparison of generated α-vectors for full VI and PBVI for tiger example after 30 iterations (from slides of M. Herrmann, RL 13).

Point-based updates and point-based value iteration (PBVI)

Let $\mathcal{B} = \{b^1, b^2, \ldots\}$ be a set of $|\mathcal{B}|$ belief points. Point-based value iteration performs Bellman update only for this limited set of belief points:

- the set of belief points $\mathcal{B}$ can correspond to a uniform coverage of the belief space or the points can focus on more relevant parts of the belief space



Characteristics of PBVI

Advantages of PBVI:

- removes exponential complexity (the number of alpha vectors is bounded)
- a practical algorithm for solving POMDPs

Disadvantages of PBVI:

- it is not clear how far from the optimum is the current solution
- the set of belief points needs to be updated / maintained
- it is not clear which part of the belief space to explore

Heuristic Search Value Iteration (HSVI)

Approximates the value function with 2 approximate value functions:

- lower bound – a set of alpha vectors corresponding to infinite-step policies
- upper bound – a set of points overestimating values for each belief point

Steps of the algorithm:

1. initialization of lower bound and upper bound approximate functions
2. selecting belief points to update using a forward search (selecting the best action to explore most promising space of belief points)
3. performing point-based updates for both approximate functions

Heuristic Search Value Iteration (HSVI)

Question

How to initialize lower / upper bound value function approximations?

- lower bound – choosing some action in all belief points all the time is clearly a lower bound on the expected reward
- upper bound – solving a simplified problem $\rightarrow$ solving an MDP for each state $s \in \mathcal{S}$

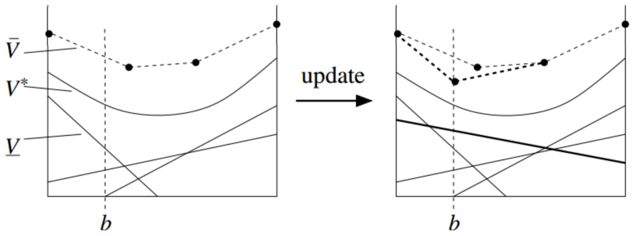
Updates:

- lower bound – point-based value updates (only the set of belief points is not bounded)
- upper bound – compute a lower convex envelope of a set of points in the upper bound and then use point-based value updates

Heuristic Search Value Iteration (HSVI)

Updates:

- lower bound – point-based value updates (only the set of belief points is not bounded)
- upper bound – compute a lower convex envelope of a set of points in the upper bound and then use point-based value updates



After an update, a new α vector is added into the lower bound and/or a new point is added into the upper bound.

Heuristic Search Value Iteration (HSVI)

Selection of the belief points to explore:

- the algorithm explores the most promising actions $\rightarrow$
  - the algorithm selects the action based on the upper bound approximation
  - see the connection with search-based methods $\rightarrow$ the upper bound is an optimistic evaluation of each belief point
  - the idea is either to (1) prove that the most-promising action actually leads to this reward (thus increase the lower bound) or (2) prove that the reward was overestimated and thus decrease the upper bound for relevant belief points
- the updates for the same action is performed for lower bound approximation

This is a very common structure of AI algorithms – upper bound drives the search, lower bounds maintains the best-found solution.

Scaling up – Monte Carlo for POMDPs

Monte Carlo Tree Search (MCTS) methods were discussed in the context of two player games.

However, we can use the same ideas for solving MDPs and also POMDPs $\rightarrow$ **POMCP** algorithm.

Question

How to construct a Monte Carlo tree?

In games (and also for MDPs), there are perfectly observable states (histories of actions) where a bandit algorithm (UCB) is used. But states are not observable in POMDPs ...

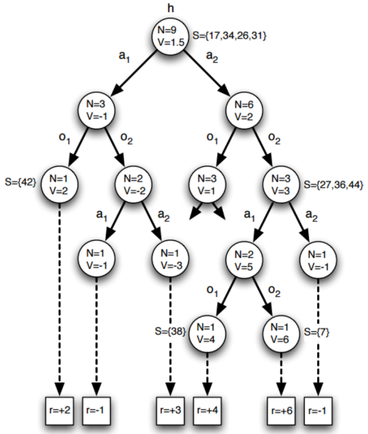
Instead, we can use **action-observation histories**.

- the agent is starting in some initial belief
- executing some action generates possible observations (according to a known probability distribution)
- receiving observation updates belief in a well-defined manner (computing a belief update)

POMCP – Monte Carlo for POMDPs

Why action-observation histories are sufficient?

- the agent is starting in some initial belief
- executing some action generates possible observations (according to a known probability distribution)
- receiving observation updates belief in a well-defined manner (computing a belief update)



POMCP – Monte Carlo for POMDPs

Where is the catch?

Bayes update of belief points can be too computationally expensive for large domains (with many states and/or observations).

The belief update can be done using **particle filtering**:

- execute K random trials (randomly choosing a true world state based on current belief, executing an action, determining the next state)
- this way we can approximate the next belief points (with probabilities of receiving an observation)

HSVI vs. POMCP (exact vs. sampling-based)

Question

What are the advantages / disadvantages of exact/approximate algorithm compared to a sampling-based approach?

- exact/approximate algorithms are better (in terms of the quality of found solution) for small cases
- exact/approximate algorithms do not scale for larger domains (HSVI will suffer from increasing number of α-vectors, maintaining the upper bound approximate function)
- sampling-based methods are in theory capable of producing a reasonable solution even for large problems
- however, often, additional improvements are necessary for sampling-based algorithms (as the particle filtering for POMCP)

POMDPs and RL

We have seen that MDPs with huge state space can be tackled by (deep) reinforcement learning algorithms.

RL-based methods can be also used for POMDPs (recall that a POMDP is “only” an MDP with infinitely large (continuous) state space).

Atari games that require memory (i.e., it is not sufficient to choose the best action only from the single (or a small number of) image(s) correspond to POMDPs rather than MDPs (there is some hidden state).

DQNs have been successfully used also for POMDPs (for example with NNs that use notion of memory – Deep Recurrent Q-Learning for Partially Observable MDPs link).

From POMDPs to Games with Imperfect Information and DeepStack

Why are imperfect information games different from single-agent (or pefect-info) problems?

1. to play optimally, the players may need to randomly choose an action from some probability distribution over available actions
   - consider, for example, rock-paper-scissors game $\rightarrow$ when playing, you randomly choose each action with probability $1/3$
   - MDPs, POMDPs, 2 player perfect info games $\rightarrow$ it is sufficient to consider only a single best action
2. in POMDPs, only the actions of the player and the well-defined environment affect the belief point $\rightarrow$ in games, there are also (possibly unobservable) actions of the opponent
   - consider, for example, a network-security problem where the attacker infects some hosts $\rightarrow$ the network admin does not directly observe these actions

From POMDPs to Games with Imperfect Information and DeepStack

What are the consequences? Why cannot we simply use RL for imperfect information games?

We can, but it is substantially more difficult to learn optimal probability distribution than to identify the best action for a decision point.

Consequently, for imperfect information games that do not require that much randomization, RL-based approach can lead to superhuman performance (for example, in Dota 2).

For other imperfect information games where randomization is crucial, simple RL-based methods do not work well (e.g., poker).

Deepstack

For poker, however, a variant of limited lookahead algorithm was designed (link) that beat professional human poker players.

DeepStack

Key components:

- instead of a belief point (possible states) of a single player, we need to consider a set of states that some of the players consider possible
- instead of evaluating a single state using a heuristic eval function (e.g., a NN), we need to evaluate a set of possible states (with probabilities)
- formulating and solving a valid limited-lookahead game requires additional steps (it is not only a simple subtree of a game tree) and a more complex algorithm

Active Research Area

In poker, the amount of unknown information is constant (opponent's cards). In real-world games, the amount of unknown information grows exponentially.

Designing an algorithm for online game playing of imperfect information is an ongoing challenge.

For games with very long horizon (or even unbounded/infinite), the history cannot be used for identifying decision points $\rightarrow$ for some games where only one player has imperfect information, HSVI algorithm can be adopted (link).

Overview of the course

We have covered basic areas of AI:

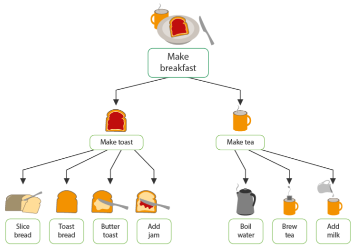
- formal representation – MDPs/POMDPs, CSP, Logic
- search – (un)informed, branch-and-bound prunning (alpha-beta), sampled-based approaches
- reinforcement learning
- dealing with uncertainty – Bayesian networks, sequential decision making under uncertainty

Described methods are general (domain-independent), the ideas can be used for solving many types of problems.

We have highlighted that typically a (novel) combination of these techniques can lead to great (groundbreaking) results.

There are new challenges ahead.

Overview of the course (2)



- AI is about problem-solving
- Real-world problems are often not solvable by a single technique
- You need to analyze and decompose, identify and solve sub-problems, combine, ...

... and read, learn, search (Google / Bing / Duck / ...)
inspiration at top AI conferences (NeurIPS, IJCAI, AAAI, ICML)
knowing the state of the art allows you progress faster