MADP Toolbox 0.1

Frans Oliehoek
faolieho@science.uva.nl

Matthijs Spaan
mtjspaan@isr.ist.utl.pt

January 18, 2008

1 Introduction

This text describes the Multiagent decision process (MADP) Toolbox, which is a software toolbox for scientific research in decision-theoretic planning and learning in multiagent systems (MASs). We use the term MADP to refer to a collection of mathematical models for multiagent planning: multiagent Markov decision processes (MMDPs) [Boutilier, 1996], decentralized MDPs (Dec-MDPs), decentralized partially observable MDPs (Dec-POMDPs) [Bernstein et al., 2002], partially observable stochastic games (POSGs) [Hansen et al., 2004], etc.

The toolbox is designed to be rather general, potentially providing support for all these models, although so far most effort has been put in planning algorithms for discrete Dec-POMDPs. It provides classes modeling the basic data types of MADPs (e.g., action, observations, etc.) as well as derived types for planning (observation histories, policies, etc.). It also provides base classes for planning algorithms and includes several applications using the provided functionality. For instance, applications that use JESP or brute-force search to solve `.dpomdp` files for a particular planning horizon. In this way, Dec-POMDPs can be solved directly from the command line. Furthermore, several utility applications are provided, for instance one which empirically determines a joint policy's control quality by simulation.

This document is split in two parts. The first presents a mathematical model of the family of MADPs, which also introduces notation, gives a high-level overview of the toolbox and an example of how to use it. In the second part, some more specific design choices and mechanisms are explained.

MADPs and overview

2 MADPs and basic notation

As mentioned, MADPs encompass a number of different models. Here we introduce the components of these mathematical models and some basic notation.

2.1 Discrete time MASs

A MADP is often considered for a particular finite number of discrete time steps t. When searching policies (*planning*) that specify h actions, this number

is referred to as the (planning-)*horizon*. So typically we look at time steps:

$$t = 0, 1, 2, ..., h-2, h-1.$$

At each time step:

- The world is in a specific state $s \in \mathcal{S}$.
- Each agent receives an individual observation: a (noisy) observation of the environment's state.
- The agents take an action.

The individually selected actions form a joint action. After such a joint action, the system jumps to the next time step. In this jump the systems' state may change stochastically, and this transition is influenced by the taken joint action. In MADP models (such as the Dec-POMDP), there are transition and observation functions describing the probability of state transitions and observations.

2.2 Basic MADP components

More formally, a multiagent decision process (MADP) consists of a subset of the following components:

- A finite set of n agents.
- $\mathcal{S}$ is a finite set of world states.
- The set $\mathcal{A} = \times_i \mathcal{A}_i$ is the set of *joint actions*, where $\mathcal{A}_i$ is the set of actions available to agent i. Every time step, one joint action $\mathbf{a} = \langle a_1, ..., a_n \rangle$ is taken. Agents do not observe each other's actions.
- $\mathcal{O} = \times_i \mathcal{O}_i$ is the set of joint observations, where $\mathcal{O}_i$ is a finite set of observations available to agent i. Every time step one joint observation $\mathbf{o} = \langle o_1, ..., o_n \rangle$ is received, from which each agent i observes its own component o_i.
- $b^0 \in P(\mathcal{S})$, is the initial state distribution at time $t = 0$.[1]
- A transition function that specifies the probabilities $P(s'|s, \mathbf{a})$.
- An observation function that specifies the probabilities $P(\mathbf{o}|\mathbf{a}, s')$.
- A set of reward functions $R(s, \mathbf{a})$ that specify the payoffs of the agents.

The partially observable stochastic game (POSG) is the most general model in the MADP-family. Dec-POMDPs are similar, but all the agents receive the same reward, so only 1 reward function is needed.

Unless stated otherwise, we use superscript for time-indices. I.e., a_i^2 denotes the agent i's action at time $t = 2$.

[1] We use $P(X)$ to denote the infinite set of probability distributions over the finite set X.

2.3 Histories

Let us more formally consider what the history of the process is. A MADP history of horizon h specifies h time-steps $t = 0, ..., h-1$. At each of these time-steps, there is a state s^t, joint observation $\mathbf{o}^t$ and joint action $\mathbf{a}^t$. Therefore, when the agents will have to select their k-th actions (at $t = k-1$), the history of the process is a sequence of states, joint observations and joint actions, which has the following form:

$$\left(s^0, \mathbf{o}^0, \mathbf{a}^0, s^1, \mathbf{o}^1, \mathbf{a}^1, ..., s^{k-1}, \mathbf{o}^{k-1}\right).$$

Here s^0 is the initial state, drawn according to the initial state distribution b^0. The initial joint observation $\mathbf{o}^0$ is usually assumed to be the empty joint observation: $\mathbf{o}^0 = \mathbf{o}_\emptyset = \langle o_{1,\emptyset}, ..., o_{n,\emptyset} \rangle$. Consequently in the MADP toolbox there is no initial observation.

An agent can only observe his own actions and observations. Therefore we introduce notions of histories from the perspective of an agent. We start with the *action-observation history* of agent i at time step t:

$$\vec{\theta}_i^t = \left(a_i^0, o_i^1, a_i^2, ..., o_i^{t-1}, a_i^{t-1}, o_i^t\right)$$

note that the choice points for the agents are right before the action:

$$\vec{\theta}_i^k = \left(\underset{\uparrow t=0}{a_i^0, o_i^1}, \underset{\uparrow t=1}{a_i^2, ..., o_i^{k-1}}, \underset{\uparrow t=k-1}{a_i^{k-1}, o_i^k} \right)$$

Therefore, when we write

$$\vec{o}_i^t = \left(o_i^1, , ..., o_i^{t-1}, o_i^t\right)$$

for agent i's *observation history* at time step t and

$$\vec{a}_i^t = \left(a_i^0, a_i^1, , ..., a_i^{t-1}\right)$$

for the *action history* of agent i at time step t, we can redefine the action-observation history as:

$$\vec{\theta}_i^t \equiv \left\langle \vec{o}_i^t, \vec{a}_i^t \right\rangle.$$

For time step $t = 0$, we have that

- $\vec{a}_i^0 = (()) = \vec{a}_\emptyset$
- $\vec{o}_i^0 = (()) = \vec{o}_\emptyset$

are empty sequences.

3 Overview of the MADP Toolbox

The framework consists of several parts, grouped in different libraries. These are briefly discussed here.

3.1 The base library (libMADPBase)

The base library is the core of the MADP toolbox. It contains:

- A representation of the basic elements in a decision process such as states, (joint) actions and observations.
- A representation of the transition, observation and reward models in a multiagent decision process. These models can also be stored in a sparse fashion.
- A uniform representation for MADP problems, which provides an interface to a problem's model parameters.
- Auxiliary functionality regarding manipulating indices, exception handling and printing: `E`, `IndexTools`, `PrintTools`. Some project-wide definitions are stored in the `Globals` namespace.

3.2 The parser library (libMADPParser)

The parser library only depends on the base library, and contains a parser for `.dpomdp` files, which is a file format for problem specifications of discrete Dec-POMDPs. A set of benchmark problem files can be found in the problems/ directory, and the `.dpomdp` syntax is documented in `problems/example.dpomdp`. The format is based on Tony's POMDP file format, and the formal specification is found in `src/parser/dpomdp.spirit`. The parser uses the Boost Spirit library.

3.3 The support library (libMADPSupport)

The support library contains basic data types useful for planning, such as:

- A representation for (joint) histories, for storing and manipulating observation, action and action-observation histories.
- A representation for (joint) beliefs, both stored as a full vector as well as a sparse one.
- Functionality for representing (joint) policies, as mappings from histories to actions.
- An implementation of the DecTiger problem which does not use `dectiger.dpomdp`, see `ProblemDecTiger`.
- Functionality for handling command-line arguments is provided by `ArgumentHandlers`.

3.4 The planning library (libMADPplanning)

The planning library depends on the other libraries and contains functionality for planning algorithms. In particular it contains

- Shared functionality for discrete MADP planning algorithms, collect in `PlanningUnitMADPDiscrete` and `PlanningUnitDecPOMDPDiscrete`. Computes (joint) history trees, joint beliefs, and value functions.

```
1  #include "ProblemDecTiger.h"
2  #include "JESPExhaustivePlanner.h"
3  int main()
4  {
5      ProblemDecTiger dectiger;
6      JESPExhaustivePlanner jesp(3,&dectiger);
7      jesp.Plan();
8      cout << jesp.GetExpectedReward() << endl;
9      cout << jesp.GetJointPolicy()->SoftPrint() << endl;
10     return(0);
11 }
```

Figure 1: A small example program that runs JESP on the DecTiger problem.

- Two Dec-POMDP solution algorithms: `JESPExhaustivePlanner` and `BruteForceSearchPlanner`. They extend the planning units mentioned before.
- A simple simulator to empirically test the control quality of a solution, see `SimulationDecPOMDPDiscrete`.

4 Using the MADP toolbox

Here we give an example of how to use the MADP toolbox. Figure 1 provides the full source code listing of a simple program. It uses exhaustive JESP to plan for 3 time steps for the DecTiger problem, and prints out the computed value as well as the policy. Line 5 constructs an instance of the DecTiger problem directly, without the need to parse `dectiger.dpomdp`. Line 6 instantiates the planner, with as arguments the planning horizon and a reference to the problem it should consider. Line 7 invokes the actual planning and lines 8 and 9 print out the results.

This is a simple but complete program, and in the distribution (in `src/examples`) more elaborate examples are provided which, for instance, demonstrate the command-line parsing functionality and the use of the `.dpomdp` parser.

Background and design of particular functionality

5 Indices for discrete models

Although the design allows for extensions, the MADP toolbox currently only provides implementation for discrete models. I.e., models where the sets of states, actions and observations are discrete. For such discrete models, implementation typically manipulates indices, rather than the basic elements themselves. The MADP toolbox provides such index manipulation functions. In particular, here we describe how individual indices are converted to and from joint indices.

5.1 Enumeration of joint actions and observations

As a convention, joint actions $\mathbf{a} = \langle a_1, ..., a_n \rangle$ are enumerated as follows

$$\begin{array}{rcl} \langle 0, \ldots, 0, 0 \rangle & — & 0 \\ \langle 0, \ldots, 0, 1 \rangle & — & 1 \\ \vdots & \vdots & \vdots \\ \langle 0, \ldots, 1, 0 \rangle & — & |\mathcal{A}_n| \\ \vdots & \vdots & \vdots \\ \langle 1, \ldots, 1, 1 \rangle & — & |\mathcal{A}_1| \cdot \ldots \cdot |\mathcal{A}_n| - 1. \end{array}$$

This enumeration is enforced by `ConstructJointActions` in `MADPComponentDiscreteActions`. The joint action index can be determined using the `IndividualToJointIndices` functions from IndexTools.h. This file also lists functions for the reverse operation.

Joint observation enumeration is analogous to joint action enumeration (and therefore the same functions can be used).

5.2 Enumeration of (joint) histories

Most planning procedures work with indices of histories. For example, PolicyPureVector implements a mapping not from observation histories to actions, but from indices (of typically observation-) histories to indices (of actions).

It is important to be able convert between indices of joint/individual action/observation histories and therefore that the method by which the enumeration is performed is clear. This is what is described in this section.

The number of such histories is dependent on the number of observations for each agent, as well as the planning history h. As a result the auxiliary functions for histories have been included in `PlanningUnitMADPDiscrete`. This class also provides the option to generate and cache joint (action-) observation histories, so that the computations described here do not have to be performed every time.

5.2.1 Observation histories

Figure 2 illustrates how observation histories are enumerated. This enumeration is

- based on the indices of the observations of which they consist.
- breadth-first, such that smaller histories have lower indices and histories for a particular time step t occupy a closed range of indices (also indicated in figure 2).

We will now describe the conversion between observation history indices and observation indices in more detail.

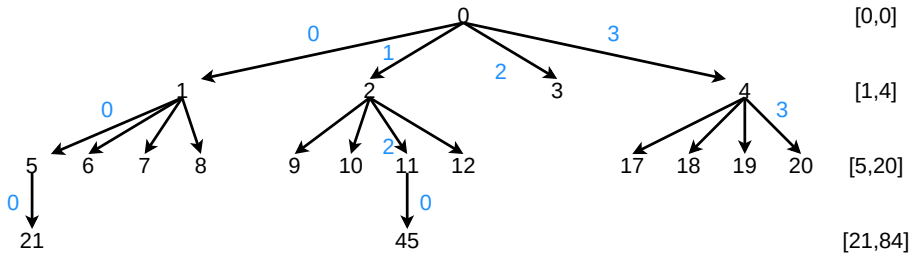


Figure 2: Illustration of the enumeration of (joint) observation histories. This illustration is based on a MADP with 4 (joint) observations.

Observation indices to observation history index In order to convert a sequence of observation indices up to time step k for agent i $\left(I_{o_i^1}, I_{o_i^2}, ..., I_{o_i^k}\right)$ [2] to an observation history index, the following formula can be used:

$$I_{\vec{o}_i^k} = \text{offset}_k + \left(I_{o_i^1} \cdot |\mathcal{O}_i|^{k-1} + I_{o_i^2} \cdot |\mathcal{O}_i|^{k-2} + ... + I_{o_i^{k-1}} \cdot |\mathcal{O}_i|^1 + I_{o_i^k} \cdot |\mathcal{O}_i|^0\right),$$

I.e., $\left(I_{o_i^1}, I_{o_i^2}, ..., I_{o_i^k}\right)$ is interpreted as a base-$|\mathcal{O}_i|$ number and offset by

$$\text{offset}_k = \sum_{j=0}^{k-1} |\mathcal{O}_i|^j - 1 = \frac{|\mathcal{O}_i|^k - 1}{|\mathcal{O}_i| - 1} - 1.$$

(One is subtracted, because the indices start numbering from 0.)

As an example, the sequence leading to index 45 in figure 2 is $(1, 2, 0)$ with $k = 3$ and $|\mathcal{O}_i| = 4$. We therefore get:

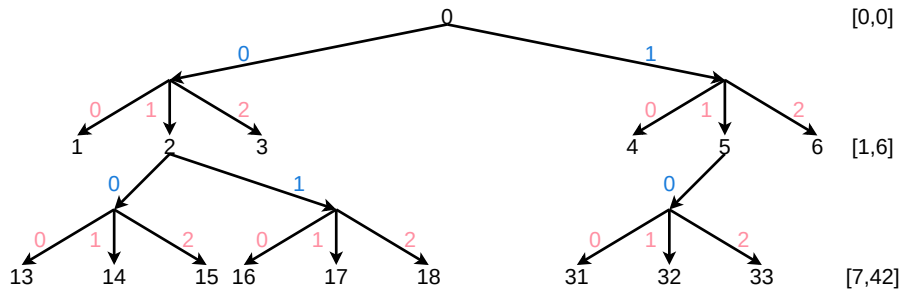
$$\left[\frac{4^3 - 1}{4 - 1} - 1\right] + \left[1 \cdot 4^2 + 2 \cdot 4^1 + 0 \cdot 4^0\right] =$$
$$\frac{64 - 1}{3} + 16 + 8 + 0 =$$
$$21 + 24 = 45.$$

This conversion is performed by `GetObservationHistoryIndex`.

Observation history index to observation indices The inverse is given by a standard division procedure:

$$\begin{aligned} 45 - 21 = 24 &\overset{\%4}{\rightarrow} 0 \\ &\overset{/4}{\rightarrow} 6 \overset{\%4}{\rightarrow} 2 \\ &\overset{/4}{\rightarrow} 1 \end{aligned}$$

[2] Note, we assume the initial observation o_i^0 to be empty. I.e. the sequence of indices $\left(I_{o_i^1}, I_{o_i^2}, ..., I_{o_i^k}\right)$ corresponds to the following sequence of observations: $\left(o_{i,\emptyset}, o_i^1, o_i^2, ..., o_i^k\right)$.



1,2,... (joint) action indices

1,2,... (joint) observation indices

1,2,... (joint) action-observation history indices

Figure 3: Illustration of the enumeration of (joint) action-observation histories. This illustration is based on a MADP with 2 (joint) actions and 3 (joint) observations.

5.2.2 Action histories

Individual action histories are enumerated exactly the same way as observation histories: a sequence of actions indices up to time step k $\left(I_{a_i^0}, I_{a_i^1}, ..., I_{a_i^{k-1}}\right)$ can be converted to an action history index by:

$$I_{\vec{a}_i^k} = \text{offset}_k + \left(I_{a_i^0} \cdot |\mathcal{A}_i|^{k-1} + ... + I_{a_i^{k-1}} \cdot |\mathcal{A}_i|^0\right).$$

5.2.3 Action-observation histories

Enumeration of action-observation histories follows the same principle as for observation histories (and action histories), but have a complicating factor. 'Action-observations' are no data type and indices are not clearly defined.

Therefore, in order to implement action-observation histories an enumeration of action-observation is assumed. Let an action observation history $\vec{\theta}_i^k = \left(o_i^0, a_i^0, o_i^1, a_i^1, o_i^2, ..., a_i^{k-1}, o_i^k\right)$ be characterized by its indices $\left(I_{a_i^0}, I_{o_i^1}, I_{a_i^1}, I_{o_i^2}, ..., I_{a_i^{k-1}}, I_{o_i^k}\right)$ (again we assume no initial observation). We can group these indices as $\left(\left\langle I_{a_i^0}, I_{o_i^1}\right\rangle, \left\langle I_{a_i^1}, I_{o_i^2}\right\rangle, ..., \left\langle I_{a_i^{k-1}}, I_{o_i^k}\right\rangle\right)$, such that each $\left\langle I_{a_i^{t-1}}, I_{o_i^t}\right\rangle$ corresponds to an action-observation. Clearly, there are $|\mathcal{A}_i| \cdot |\mathcal{O}_i|$ action-observations. Let's denote an action-observation with θ_i and its index with I_{θ_i}, corresponding to action a_i and observation o_i, we then have that:

$$I_{\theta_i} = I_{a_i} \cdot |\mathcal{O}_i| + I_{o_i}.$$

`ActionAndObservation_to_ActionObservationIndex` from `IndexTools.h` performs this computation. The inverse operation is performed by `ActionObservation_to_ActionIndex` and `ActionObservation_to_ObservationIndex`,

Now these indices are defined, the same procedure for observation histories can be used as illustrated in fig. 3. I.e.,

$$\begin{aligned} I_{\vec{\theta}_i^k} =& \text{offset}_k + I_{\theta_i^1} \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^{k-1} + I_{\theta_i^2} \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^{k-2} + ... \\ & + I_{\theta_i^{k-1}} \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^1 + I_{\theta_i^k} \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^0 , \end{aligned}$$

Note that

$$\begin{aligned} I_{\theta_i^p t} \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^{k-t} &= \left(I_{a_i^t} \cdot |\mathcal{O}_i| + I_{o_i^t} \right) \cdot (|\mathcal{A}_i| \cdot |\mathcal{O}_i|)^{k-t} \\ &= I_{a_i^t} \cdot |\mathcal{O}_i|^{k-t+1} \cdot |\mathcal{A}_i|^{k-t} + I_{o_i^t} |\mathcal{O}_i|^{k-t} \cdot |\mathcal{A}_i|^{k-t} \end{aligned}$$

As an example, index 32 is corresponds to index sequence $(1, 1, 0, 1)$ which, in action-observation indices, corresponds with $(4, 1)$ and thus:

$$\begin{aligned} \left(6^0 + 6^1\right) + 4 \cdot 6^{2-1} + 1 \cdot 6^{2-2} = \\ 7 + 24 + 1 = 32 \end{aligned}$$

Alternatively we can use the sequence $(1, 1, 0, 1)$ directly:

$$\begin{aligned} 7 + \left(1 \cdot 3^{2-1+1} \cdot 2^{2-1} + 1 \cdot 3^{2-1} \cdot 2^{2-1}\right) + \left(0 \cdot 3^{1-1+1} \cdot 2^{1-1} + 1 \cdot 3^{1-1} \cdot 2^{1-1}\right) = \\ 7 + \left(1 \cdot 3^2 \cdot 2^1 + 1 \cdot 3^1 \cdot 2^1\right) + \left(0 \cdot 3^1 \cdot 2^0 + 1 \cdot 3^0 \cdot 2^0\right) = \\ 7 + (1 \cdot 9 \cdot 2 + 1 \cdot 3 \cdot 2) + (0 \cdot 3 \cdot 1 + 1 \cdot 1 \cdot 1) = \\ 7 + 18 + 6 + 1 = \\ 7 + 25 = 32 \end{aligned}$$

5.2.4 Joint histories

Joint observations are enumerated in the same way as individual observation histories, only now using the indices of joint observations rather than individual observations.

I.e., figure 2 also illustrates how joint observation histories are enumerated. And in order to convert a sequence of joint observations indices up to time step k $(I_{\mathbf{o}^1}, ..., I_{\mathbf{o}^k})$ to an observation history index, the following formula can be used:

$$I_{\vec{\mathbf{o}}} = \text{offset}_k + \left(I_{\mathbf{o}^0} \cdot |\mathcal{O}|^{k-1} + I_{\mathbf{o}^1} \cdot |\mathcal{O}|^{k-1} + ... + I_{\mathbf{o}^{k-1}} \cdot |\mathcal{O}|^1 + I_{\mathbf{o}^k} \cdot |\mathcal{O}|^0 \right),$$

I.e., $(I_{\mathbf{o}^1}, ..., I_{\mathbf{o}^k})$ is interpreted as a base-$|\mathcal{O}|$ number and offset by

$$\text{offset}_{k+1} = \sum_{j=0}^{k-1} |\mathcal{O}|^j - 1 = \frac{|\mathcal{O}|^k - 1}{|\mathcal{O}| - 1} - 1.$$

Indices for joint action histories and joint action-observation histories are computed in the same way. The action-observation functions (`ActionObservation_to_ActionIndex`, etc.) can also be used for joint action-observations.

`PlanningUnitMADPDiscrete` also provides functions to convert joint to individual history indices `JointToIndividualObservationHistoryIndices`, etc.

6 Joint beliefs and history probabilities

Planning algorithms for MADPs will typically need the probabilities of particular joint action observation histories, and the probability over states they induce (called joint beliefs). `PlanningUnitMADPDiscrete` provides functionality for this, which we discuss here.

6.1 Theory

Let $P_\pi(\mathbf{a}^t|\vec{\boldsymbol{\theta}}^t)$ denote the probability of $\mathbf{a}$ as specified by π, then $P(s^t, \vec{\boldsymbol{\theta}}^t|\pi, b^0)$ is recursively defined as

$$P(s^t, \vec{\boldsymbol{\theta}}^t|\pi, b^0) = \sum_{s^{t-1}\in\mathcal{S}} P(s^t, \vec{\boldsymbol{\theta}}^t|s^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi)P(s^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}|\pi, b^0). \quad (1)$$

with

$$P(s^t, \vec{\boldsymbol{\theta}}^t|s^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi) = P(\mathbf{o}^t|\mathbf{a}^{t-1}, s^t)P(s^t|s^{t-1}, \mathbf{a}^{t-1})P_\pi(\mathbf{a}^{t-1}|\vec{\boldsymbol{\theta}}^{t-1}).$$

For stage 0 we have that $\forall_{s^0}\ P(s^0, \vec{\boldsymbol{\theta}}_\emptyset|\pi, b^0) = b^0(s^0)$.

6.1.1 Split

Since we tend to think in joint beliefs $b^{\vec{\boldsymbol{\theta}}^t}(s^t) \equiv P(s^t|\vec{\boldsymbol{\theta}}^t, \pi, b^0)$, we can also represent the distribution (1) as:

$$P(s^t, \vec{\boldsymbol{\theta}}^t|\pi, b^0) = P(s^t|\vec{\boldsymbol{\theta}}^t, \pi, b^0)P(\vec{\boldsymbol{\theta}}^t|\pi, b^0) \quad (2)$$

The joint belief $P(s^t|\vec{\boldsymbol{\theta}}^t, \pi, b^0)$ The joint belief $P(s^t|\vec{\boldsymbol{\theta}}^t, \pi, b^0)$ is given by:

$$\begin{aligned} P(s^t|\vec{\boldsymbol{\theta}}^t, \pi, b^0) &= \frac{P(\mathbf{o}^t|\mathbf{a}^{t-1}, s^t)\sum_{s^{t-1}} P(s^t|s^{t-1}, \mathbf{a}^{t-1})P(s^{t-1}|\vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)}{\sum_{s^t} P(\mathbf{o}^t|\mathbf{a}^{t-1}, s^t)\sum_{s^{t-1}} P(s^t|s^{t-1}, \mathbf{a}^{t-1})P(s^{t-1}|\vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)} \\ &= \frac{P(s^t, \mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)}{P(\mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)} \end{aligned} \quad (3)$$

where $P(\mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0) = P(\vec{\boldsymbol{\theta}}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)$.

The probability of an history $P(\vec{\boldsymbol{\theta}}^t|\pi, b^0)$ The second part of (2) is given by

$$\begin{aligned} P(\vec{\boldsymbol{\theta}}^t|\pi, b^0) &= P(\vec{\boldsymbol{\theta}}^t|\vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)P(\vec{\boldsymbol{\theta}}^{t-1}|\pi, b^0) \\ &= P(\vec{\boldsymbol{\theta}}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)P_\pi(\mathbf{a}^{t-1}|\vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)P(\vec{\boldsymbol{\theta}}^{t-1}|\pi, b^0) \\ &= P(\mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)P_\pi(\mathbf{a}^{t-1}|\vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)P(\vec{\boldsymbol{\theta}}^{t-1}|\pi, b^0) \quad (4) \end{aligned}$$

where $P(\mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\boldsymbol{\theta}}^{t-1}, \pi, b^0)$ is the denominator of (3).

Algorithm 1 $[b^{\vec{\theta}^t}, P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}})] =$ `GetJAOHProbs`$(\vec{\theta}^t, \pi, b^{\vec{\theta}^{t'}}, \vec{\theta}^{t'})$

1: **if** $\vec{\theta}^t = \vec{\theta}^{t'}$ **then**
2: **return** $[b^{\vec{\theta}^t} = b^{\vec{\theta}^{t'}},\ P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}}) = 1\ ]$
3: **end if**
4: **if** $\vec{\theta}^t$ not an extention of $\vec{\theta}^{t'}$ **then**
5: **return** $[b^{\vec{\theta}^t} = \vec{0},\ P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}}) = 0\ ]$
6: **end if**
7: $\vec{\theta}^{t''} = (\vec{\theta}^{t'}, \mathbf{a}^{t'}, \mathbf{o}^{t'+1})$ {consist. with $\vec{\theta}^t$}
8: $[b^{\vec{\theta}^{t''}}, P(\vec{\theta}^{t''}|\mathbf{a}^{t'}, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}})] = b^{\vec{\theta}^{t'}}$`.Update`$(\mathbf{a}^{t'}, \mathbf{o}^{t'+1})$ {belief update, see (3)}
9: $[b^{\vec{\theta}^t}, P(\vec{\theta}^t|\pi, \vec{\theta}^{t''}, b^{\vec{\theta}^{t''}})] =$ `GetJAOHProbs`$(\vec{\theta}^t, \pi, b^{\vec{\theta}^{t''}}, \vec{\theta}^{t''})$
10: $P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}}) = P(\vec{\theta}^t|\pi, \vec{\theta}^{t''}, b^{\vec{\theta}^{t''}}) P(\vec{\theta}^{t''}|\mathbf{a}^{t'}, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}}) P_\pi(\mathbf{a}^{t'}|\vec{\theta}^{t'}, \pi)$
11: **return** $[b^{\vec{\theta}^t},\ P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}})\ ]$

6.2 Implementation

Since computation of $P(\vec{\theta}^t|\pi, b^0)$ is interwoven with the computation of the joint belief through $P(\mathbf{o}^t|\mathbf{a}^{t-1}, \vec{\theta}^{t-1}, \pi, b^0)$, it is impractical to separately evaluate (3) and (4).

Rather we define a function

$$[b^{\vec{\theta}^t}, P(\vec{\theta}^t|\pi, b^0)] = \texttt{GetJAOHProbs}(\vec{\theta}^t, \pi, b^0)$$

Because in many situations an application might evaluate similar $\vec{\theta}^t$ (i.e., ones with an identical prefix), a lot of computation will be redundant. To give the user the possibility to avoid this, we also define

$$[b^{\vec{\theta}^t}, P(\vec{\theta}^t|\pi, \vec{\theta}^{t'}, b^{\vec{\theta}^{t'}})] = \texttt{GetJAOHProbs}(\vec{\theta}^t, \pi, b^{\vec{\theta}^{t'}}, \vec{\theta}^{t'})$$

which returns the probability and associated joint belief of $\vec{\theta}^t$, *given* that $\vec{\theta}^{t'}$ (and associated joint belief $b^{\vec{\theta}^{t'}}$) are realized (i.e., given that $P(\vec{\theta}^{t'}) = 1$).

7 Policies

Here we discuss some properties and the implementation of policies. Policies are plans for agents that specify how they should act in each possible situation. As a result a policy is a mapping from these 'situations' to actions. Depending on the assumption on the observability in a MADP, however, these 'situations' might be different. Also we would like to be able to reuse the implementations of policies for problems with a slightly different nature, for instance (Bayesian) games.

In the MADP toolbox, the most general form of a policy is a mapping from a domain (`PolicyDomain`) to (probability distributions over) actions. Currently we have only considered policies for discrete domains, which are mappings from indices (of these histories) to indices (of actions). It is typically still necessary to know what type of indices a policy maps from in order to be able to reuse our implementation of policies. To this end a discrete policy maintains its `IndexDomainCategory`. So far there are four types of index-domain categories: `TYPE_INDEX`, `OHIST_INDEX`, `OAHIST_INDEX` and `STATE_INDEX`.

As said `PolicyDiscrete` class represents the interface policies for discrete domains. `PolicyDiscretePure` is the interface for a pure (deterministic) policy. A class that actually implements a policy is `PolicyPureVector`. This class also implements a function to get and set the index of the policy (pure policies over a finite domain are enumerable). Joint policies are are represented by similarly named classes `JointPolicyDiscrete`, `JointPolicyPureVector`, etc.

In order to instantiate a (joint)policy, it needs to know several things about the problem it is defined over. We already mentioned the index domain category, but there is information needed as well (the number of agents, the sizes of their domains, etc.). To provide this information, each problem for which we want to construct a (joint) policy has to implement the `Interface_ProblemToPolicyDiscretePure`.

References

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Math. Oper. Res.*, 27(4):819–840, 2002. ISSN 0364-765X. doi: http://dx.doi.org/10.1287/moor.27.4.819.297.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK '96: Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1-55860-417-9.

Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI '04: Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715. AAAI Press / The MIT Press, 2004. ISBN 0-262-51183-5.