POLITECNICO DI TORINO
Master of Science in Computer Engineering

Master's Thesis

Solving Interactive POMDPs in Julia

Supervisor
Prof. Elio Piccolo

Candidate
Iacopo Olivo

December 2018

First and foremost, I would like to thank my advisors Piotr Gmytrasiewicz and Elio Piccolo, for being always available and patient in answering my doubts and questions and to help me solving problems encountered in this journey.

I also wish to acknowledge my gratitude to my parents, who gave me the opportunity to be here. They have been my number one supporters throughout my entire journey towards this degree, always believing in me, also in those times when I was the first one who did not.

I wish to thank all the friends who cheered for me during my path from my small hometown of Creazzo to this huge metropolis of Chicago, from my hometown friends to all the people from Turin. A big thank you goes to Chiara and Top del TOP-UIC friends who shared with me this period full of experiences and emotions making it an even more sensational phase of my life. A final thank you goes to all those people who, by their support, filled me with determination and positivity.

IO

Summary

Context

Stochastic partially-observable sequential environments are a good representation of the reality we live in and as a consequence, this branch is given a growing attention from the artificial intelligence community. In order to solve these problems, several frameworks have been proposed and one of the most famous is the Partially Observable Markov Decision Process (POMDP).

Interactive Partially Observable Markov Decision Process framework has been proposed in order to extend POMDPs to a multi-agent environment and it is currently undergoing a growing interest from the community. The strength of the framework is given by its ability to represent the other agents and to infer the probability of their actions.

In I-POMDPs other agents are included in the belief state, and as a consequence, the framework is able to understand the model of the agents it is interacting with.

This work contains two major themes - solving Interactive POMDPs and defining agents able to work with the framework - that alternate during both the introduction phase and the method phase of the thesis.

Agents are defined by means of a structured interface used in order to enable the communication among them. The implemented inner agent model is then used in order to act optimally in the environment. However, there is no current standard to implement agents using Interactive POMDPs as an inner model. This lack of infrastructure results in a lack of comparison and access to existent work by the research teams interested in this new framework.

Goals

The goals of this work are double: *i)* define a common interface in order to formalize agents capable of implementing IPOMDP as inner agent models *ii)* define a method in order to solve I-POMDPs in an efficient way.

The first goals include a first phase consisting of understanding the solutions already present in order to define agent for stochastic partially-observable sequential environments and understand whether it is possible to extend or include them in order to implement I-POMDPs. The second phase is the definition of the interface itself, by understanding the motivations behind other designs and extending them in order to be able to implement agents capable of acting in multi-agent settings by means of Interactive POMDPs.

The second goal is differentiated by the first due to its nature: rather than being implementation oriented it needs to solve the complexity problem for Interactive POMDPs.

It is important due to the number of states the problem can generate through time. Both phases are aimed to provide the community with a structured method to both define agents implementing Interactive POMDPs and develop solvers for the framework. Having this structure available could decrease the time needed in order to develop and implement new solvers and methods, hence speeding up the research process.

Methods and Results

Due to the nature of the goals, they are intrinsically dependent. In order to design a working interface for defining agents, it is needed a solver capable of resolving the agent function, while in order to define such solver an interface needs to be implemented.

As first, an analysis of the possible solving methods has been performed in order to define the solution method. The possibility of reducing I-POMDPs (due to their tight similarity to POMDPs) has been considered the most promising approach. After choosing the approach, the different parts of the I-POMDP framework have been derived starting from the elements of the POMDPs and generalizing them following probability theory.

In order to fulfill the interface definition goal, an analysis of the present solving methods has been performed. A very complete framework, Julia.POMDPs is already present and provides most of the characteristics needed (extensibility, ease of use, speed) and shows an interesting design for defining single-agent POMDPs. The resulting I-POMDP interface has been called Julia.IPOMDPs and takes advantage of Julia.POMDPs framework in order to perform the reduction from an Interactive to a normal POMDP. Julia.IPOMDPs deeply extends Julia.POMDPs by implementing the concept of model, an entity capable of returning optimal actions and probability distributions over its decision. The model interface allows Julia.IPOMDPs to be general enough to be used to define not only I-POMDP agents but virtually any agent which follows the classic perceive-think-act loop. Once Julia.IPOMDPs interface is defined, it is possible to implement the reduction algorithm derived in the first phase.

The last phase of the work consists in testing both Julia.IPOMDPs and the reduction solver. A set of problems have been derived from the ones present in the literature in order to create a set of examples. For the examples tested (different on the nesting level and the number of models emulated), the solution time is reported. Interesting patterns are found and the effect of other agent actions are evaluated.

The results underline how the course of history affects the run-time by means of the size of the interactive state set. Solutions are proposed to address future research that might address the problem.

Contents

List of Tables

List of Figures

Chapter 1

Motivation and Previous work

The Interactive Partially Observable Markov Decision Process framework is developed to deal with partially observable stochastic domains where more than one agent is present in the environment. The possibility of solving Interactive Partially Observable Markov Decision Processes is limited by two main factors: computational and space complexity. In order to produce solutions which can deal with these two factors, every research team needs to develop its personal development and testing environment. However, when a standardized procedure to define such problems is present, it allows be able to compare performances on particular algorithms produced by different research teams. The aim of this work is to define and propose a standardized way to describe I-POMDPs.

Interactive Partially Observable Markov Decision Process is a framework developed by expanding Partially Observable Markov Decision Process in order to include multiple agents in the environment. It is experiencing a growing interest in the Artificial Intelligence community. As a consequence of this growing involvement, examples have been developed. They are very varied and range from money laundering applications [12] to models in order to define trustworthiness of agents [18]. Further examples of applications can be found at [22].

Solving Interactive Partially Observable Markov Decision Processes is a very complex problem and requires a significant amount of resource due to the course of history and course of dimensionality as will be later explained in 2.2.2. I-POMDP has been formalized in 2004 [7] and it is a relatively new framework. In order to demonstrate its capabilities and provide the first examples, it is needed to develop new solving algorithms. Currently, several methods have been proposed:

- The *interactive particle filter* (I-PF) method explored in [5] aims to infer the possible actions of the other agents by sampling their belief state. Through the interactive particle filter algorithm, some particles are selected in order to represent the other agents' beliefs b^{t-1}. These obtained particles are then projected forward in time in order to sample the future possible belief states and consequently estimate the other agent's belief b^t. In case the other agent is an I-POMDP itself, in order to project the particle in the future the I-PF function needs to be called recursively for each nesting level of the models, until level-1, where the action for the nested level-0 models can be inferred through the use of normal POMDP belief update. All these particles need to be weighted in order to be effective. The weighting factor is the probability of receiving the observation which generated the particle given the actions of all the agents and the current interactive state.
- Value iteration is the most classical algorithm used in order to solve sequential

decision-making problems and it has been proved to be optimal for POMDPs. However, while it has been successfully applied to I-POMDPs, it has not been proved to be optimal due to the fact that Interactive Partially Observable Markov Decision Process might be self-referencing [17]. Value iteration, however, is proved to converge [7].

- Policy iteration is another classical algorithm. It has been adapted to Interactive Partially Observable Markov Decision Process in [21].

Value iteration and Policy iteration are often used along with an interactive particle filter. The former methods are used in order to control the value estimation of the states of the model, while the latter is often used in order to solve the models of the other agents which are part of the Interactive Partially Observable Markov Decision Process type.

Another interesting application of Interactive Partially Observable Markov Decision Processes is to learn other agents models as explained in [9]. The agent is demonstrated to be able to learn the models of other agents by applying Bayesian inference and sequential Monte Carlo sampling. The example is given use interactive particle filter as I-POMDP solving method.

Chapter 2

Background

Sequential decision making is the branch of artificial intelligence which deals with problems using a procedural approach and where earlier decisions influence the later state of the world. There are two important characteristics of the environment which make it particularly difficult to deal with[16]:

- *Partial observability*: An environment is said partially observable when the agent is not given access to each state of the environment for each point in time.
- *Stochasticity*: an environment is said stochastic when the outcome of the agent action is not deterministic. This can happen because of either partial observability or for the complexity of the environment due to variables and other agent presence.

Partially Observable Markov Decision Processes are found to deal particularly well in *partially observable stochastic domains* (*POSD*). However, they are not defined for multi-agent systems. Multi-agent systems are all those environments where more than one agent is present. They are extremely common since they can be used in order to express the forms of interaction used in our society. Behavior of agents [23] may be summarized in:

- *Cooperative*: agents work together in order to achieve a result
- *Competitive*: agents work against each other
- *Neutral*: agents do not really care about each other

Since the environment is very similar to POSD, it comes naturally to expand the Partially Observable Markov Decision Process framework to a multi-agent setting. In fact, there have been several trials to expand POMDP to multi-agent settings. Depending on the type of problem, several frameworks are available[23]:

- *Cooperative*: Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [13] where all the agents share the same reward function hence suitable for cooperative games.
- *Competitive*: Interactive Partially Observable Markov Decision Process (I-POMDP)[7] is a framework capable of empowering the agents with a theory of mind of the adversaries, hence suitable for competitive games
- *Indifferent*: Partially observable stochastic games [10] is an extension of stochastic games [19]

This work is focused on I-POMDPs due to its expressive power and range of implementations it can perform [4]. The ability to model the other agent's behavior makes I-POMDP suitable for all three the stated categories. In order to be able to fully understand the capabilities if Interactive Partially Observable Markov Decision Processes it is, however, necessary to introduce Partially Observable Markov Decision Processes first.

2.1 POMDP

Partially Observable Markov Decision Process is a very known framework in the AI community. It is aimed to solve single-agent POSDs hence considering the idea that the agent might not have full access to all the world states. An agent which agent function is described as a POMDP will implement:

$$POMDP = \langle S, A, T, \Omega, O, R \rangle$$

Where S is the set of all the possible states of the world and Ω are all the possible observations the agent can receive from the environment. In order to provide the agent with a complete model of the world the *Transition* and *Observation* functions are defined. The reward function describes the agent behavior.

The *Transition function* describes how the agent's actions affect the world. It is a distribution over states and actions where $\sum_{s^t \epsilon S} T(s^{t-1}, a^{t-1}, s^t) = 1$. In the special case when the transition is deterministic $T(s^{t-1}, a^{t-1}, s^t) = 1$ for the resulting s^t and 0 for all the others.

Due to the fact that the world is usually non-deterministic, we use the *Observation function* to describe the likelihood of receiving an observation by performing a certain action and arriving in a selected state. It is used in order to understand the feedback received from the environment to the agent action. Similarly for the transition function, also the observation function is a probability distribution where $\sum_{o^t \epsilon \Omega} O(a^{t-1}, s^t, o^t) = 1$. In case of deterministic observations, certain combinations will lead to $O(a^{t-1}, s^t, o^t) = 1$ for the resulting o^t and 0 for all the others. Note that in fully observable environments there is no need for specific observations. Due to the fact that the agent has access to the states of the world, we can map the observations to the states, de facto converting a Partially Observable Markov Decision Process to a Markov Decision Process.

Another important element of the POMDP framework is the *Reward function.* The reward signal is provided to the user by the environment and it is used as part of the value function the agent is trying to maximize. The reward function maps the states an actions to the reward signal and is expressed as $R(a, s)$.

Since Partially Observable Markov Decision Processes are suitable for non-deterministic environments, the agent does not track one single actual state. Instead, it keeps a distribution over all the possible states called *belief.* The belief of an agent is a probability distribution over the whole state space S and indicates the likelihood an agent is in a certain state s. When the agent is in the first phase of the environment, the belief distribution b_0 is created and indicates intuitively the initial belief of the agent. However, the agent is going to perform actions on the environments and receive both observations and rewards from it. This changes the real state of the world (which is unknown to the agent) and the belief of the agent is not considered to be updated anymore. As a consequence it is useful to define a *belief update* function. The update phase is performed when the agent performs the action a^{t-1} and receives the observation o^t:

$$b^t(s) = \alpha O(a^{t-1}, s^t, o^t) \sum_{s^{t-1} \epsilon S} b^{t-1}(s^{t-1}) T(s^{t-1}, a^{t-1}, s^t) \tag{2.1}$$

where α is used as normalization factor. The update function is also called the *State Estimation* function $SE(b^{t-1}, a^{t-1}, o^t)$ which intuitively returns the updated belief state b^t.

2.1.1 solving POMDPs

During the agent loop, the agent needs to select an action. This action aims to maximize the reward over time. In order to define the concept of the utility function, it is first useful to introduce the concept of *optimality criterion OC*. The optimality criterion indicates how are weighted the received rewards over time. By defining the reward at current time t as r_t, common criteria are:

- *Finite horizon criterion*: the parameter T (called *length of the horizon*) indicates the maximum number of future rewards r_t considered, maximizing the sum of expected rewards $E(\sum_{t=0}^{T} r_t)$
- *Infinite horizon criterion with discount*: the parameter $0 \leq \gamma \geq 1$ (called *discount factor*) indicates how influent are the future rewards r_t. The agent is hence maximizing the function $E(\sum_{t=0}^{\infty} \gamma^t r_t)$

The latter optimality criterion is used in the widely popular *Bellman optimality equation*, which describes the utility of a specific belief state:

$$
\begin{aligned}
V(b) &= \max_{a\epsilon A}[\sum_{s\epsilon S} R(s,a)b(s) + \sum_{o\epsilon\Omega}\sum_{s\epsilon S} O(s,a,o)b(s)V(SE(b,a,o))] \\
&= \max_{a\epsilon A}[R(b,a) + \sum_{o\epsilon\Omega} O(o|b,a)V(SE(b,a,o))]
\end{aligned}
\tag{2.2}
$$

As we can note, the Bellman equation can be split into two parts:

- $\sum_{s\epsilon S} R(s,a)b(s)$ is the *immediate reward* for performing the action a in the current belief state b
- $\sum_{o\epsilon\Omega}\sum_{s\epsilon S} O(s,a,o)b(s)V(SE(b,a,o))$ indicates the *discounted reward* obtainable in from the future actions

It is intuitive that, in order to obtain the second term, an eventual solver needs to branch in the future considering all the possible actions and observations. This increases significantly the complexity of POMDPs.

In order to define which action needs to be taken depending on the possible belief state, one solver needs to compute a *policy* π. Whenever this policy is considered optimal it is defined as $\pi^\star$.

In order to calculate the Partially Observable Markov Decision Process policy, several methods have been developed. The programs who implement those methods are here defined solvers. They are usually divided into two categories:

- *On-line* solvers: determine the optimal policy before acting. They move all the computations to the initial planning phase. This allows the agent to run smoothly once the policy is calculated.
- *Off-line* solvers: determine the policy at run-time. The planning phase occurs during the whole agent lifetime.

Two of the more recent solving methods developed are *SARSOP* and *DESPOT*.

- *Successive Approximation of the Reachable Space under Optimal Policies* (*SARSOP*) [11] is a POMDP off-line solving algorithm which plans over the *optimally reachable*

belief spaces (which are those belief spaces reachable by applying optimal policies) to increase computational efficiency.
- *Determinized Sparse Partially Observable Tree* (*DESPOT*) [24] is an on-line POMDP solving algorithm which exploits randomly generated scenarios in order to perform planning.

2.1.2 julia.POMDPs

Julia is a programming language developed aiming to high performance in technical computing [2]. Julia provides the possibility to execute compiled code and to interact with the console, making it simple to query and analyze the obtained data.

Within Julia ecosystem, *Julia.POMDPs* [6] is an open-source package developed in order to support the user in defining problems, running experiments and creating solvers with the aim of both encouraging the growth of its package ecosystem and the creation of new and more efficient algorithms. The main design criteria followed in the Julia.POMDPs development are *Expressiveness* of the problem definition interface, *Extensibility* of the framework in order to allow algorithms to be easily implemented within Julia.POMDPs and *usability* to allow the user to use the package with all the already existent solvers.

2.2 Interactive-POMDP

Interactive Partially Observable Markov Decision Process is a framework applicable to self-interested autonomous agents participating in a multi-agent game in a non-deterministic environment. Agents defined with I-POMDP are capable of defining advanced constructs in order to model and predict the behavior of the other agents present in the game. The Interactive Partially Observable Markov Decision Process approach is based on computing the optimal action by anticipating the response of the other agents.

In I-POMDP agents are defined based on their type and frame. A *type* of an agent is the tuple

$$\theta_i = \langle b_i, A_i, \Omega_i, T_i, R_i, OC_i \rangle \tag{2.3}$$

Where all the parameters of the tuple are described in 2.1 with OC_i indicating the optimality criterion the agent I uses in order to calculate the expected cumulative reward. The frame of an agent is defined as

$$\widehat{\theta}_i = \langle A_i, \Omega_i, T_i, R_i, OC_i \rangle \tag{2.4}$$

which is practically identical to the type definition, omitting the interactive belief distribution $b_i = \Delta(S)$. Consequently the type of an agent can be expressed as

$$\theta_i = \langle b_i, \hat{\theta}_i \rangle$$

Interactive Partially Observable Markov Decision Process generalized Partially Observable Markov Decision Process in order to include the presence of other agents. In IPOMDP notation, the other agents' presence is included in the state space, which concept is expanded in order to generate the *interactive state space*. An Interactive-POMDP of an agent I can be described as:

$$I\text{-}POMDP_i = \langle IS_i, A, T_i, O_i, \Omega_i, R_i, OC_i \rangle \tag{2.5}$$

Considering an ideal game where N agents are playing, $A = A_i \times A_j \times \cdots \times A_n$ is the set containing all the possible combinations of action they can perform. Each agent, however will be defined by its own agent type, meaning that in the case agent J is a IPOMDP, it will be defined by $I\text{-}POMDP_j = \langle IS_j, A, T_j, O_j, \Omega_j, R_j, OC_j \rangle$. It is easy to conclude that, while the action space is shared among all the IPOMDP agents playing in the game, their observation space is not. In fact, the observation space of agent I will be Ω_i which is not necessarily related to any other agent's observation space.

IS_i is called *interactive state set* of agent I. One of the characteristics of Interactive Partially Observable Markov Decision Process framework is to allow the agent to define constructs capable to model the other agents acting in the same environment in order to be able to predict their actions. This characteristic is included in the world state representation from the agent I. By defining S as the set containing all the possible states of the world and M_x the set containing all the possible models of an agent X:

$$IS_i = S \times M_j \times \cdots \times M_n \tag{2.6}$$

Similarly, the interactive belief of an Interactive Partially Observable Markov Decision Process is defined as:

$$b_i = \Delta(IS_i) \tag{2.7}$$

It is important to note that the set IS_i is infinite, due to the fact that M_j is infinite itself. A *model* of an agent J is defined as $m_j \epsilon M_j$ and it is the construct that enhances the agent I with a representation of the other agents playing in the game. Each other agent will be modeled in I's interactive state space as a model. A model is the tuple

$$m_j = \langle h_j, f_j, O_j \rangle \tag{2.8}$$

Where h_j is the *history of observations* the model received during its lifetime, f_j is the *agent function*, meaning the function which maps the model's history to its actions $f_j(h_j) \rightarrow A_j$. The last element is the *model observation function*, which indicates how the world is providing the model m_j with its observation. The term *model* and *type* of an agent look very similar but are actually different. Taking, for example, an agent J who is represented as an $I\text{-}POMDP_j$ in I's interactive state space IS_i, there are as many models m_j as the different beliefs $b_j(IS_j)$ that can be generated. As a consequence it is useful to define the model as a combination of history and frame of an agent:

$$m_j = \langle h_j, \widehat{m}_j \rangle$$

It is important to note that the number of models of an agent J is infinite, due to the fact that h_j could easily be a Kleene closure h_j^*.

The *transition function* T_i indicates how agent I maps the actions of the various agents to the world he is playing in. Each combination of agent actions s and previous state results in a probability distribution

$$T_i(s^{t-1}, a^{t-1}, s^t) = P(s^t \mid s^{t-1}, a^{t-1}) \tag{2.9}$$

It is useful to indicate that the *joint action* a^{t-1} is a composition of all the actions the various agents took at a certain time $t-1$. In a world where N agents are performing actions

$a = \langle a_i, \ldots, a_n \rangle$. Another useful insight is to note how the transition functions acts on distributions over physical states $\Delta(S)$ and not on interactive states. This is due to the *Model Non-manipulability Assumption*(MNM) which indicates that agents' actions cannot change other models directly. Since this is a probability distribution $\sum_{s^t \epsilon S} T_i(s^{t-1}, a^{t-1}, s^t) = 1$ for given a^{t-1} and s^{t-1}.

The *observation function* in a Interactive Partially Observable Markov Decision Process is, like in Partially Observable Markov Decision Processes , a probability distribution over the actions

$$O_i(s^t, a^{t-1}, o_i^t) = P(o_i^t \mid s^t, a^{t-1}) = P(o_i^t \mid s^t, a_i^{t-1}, a_j^{t-1}) \tag{2.10}$$

Indicating how the world supplies the agent I with its observations. The observation function is defined over the agent I's observation set $\sum_{o_i^t \epsilon \Omega_i} O_i(s^t, a^{t-1}, o_i^t) = 1$ and maps which observations we could receive given that the agents performed a certain combination of actions and the world transitioned to a determined physical state. The observation, however, depends only on the physical state and not on the interactive state of the problem. This is due to the *Model Non-observability Assumption* (MNO), which states that it is impossible for an agent to directly observe the inner state of the other agents.

Reward is defined in a similar manner to the Partially Observable Markov Decision Process 's reward function, only it depends on the combination of actions of all the agents:

$$R_i(is, a) = R(is, a) \tag{2.11}$$

It deeply influences the behavior of the agent and maps how the reward signal received by the environment relates to the combination of actions of the agents and the current interactive state. The reward function, however, it is not affected by MNM and MNO assumptions due to the fact that shaping the reward of the agent depending on the other agent state maintains the autonomy of the agent.

OC_i is the *optimality criterion* for the agent I. It defines the horizon and the modality the received rewards r_t are considered during the time. The most common optimality criterion for IPOMDPs is the *infinite time horizon with discount* $0 \leq \gamma \geq 1$ which, similarly to POMDPs, indicates the reward as $E(\sum_{t=0}^{\infty} \gamma^t r_t)$.

2.2.1 Belief update

Analogously to Partially Observable Markov Decision Processes , an IPOMDP agent maintains a belief on the current state of the world. However, this belief is not only on the physical states of the world S, but it includes the models of the other agents in IS. In the same way to Partially Observable Markov Decision Processes , the actions of the agents and the observation an agent receives can modify what an agent I believes about this state of the world. As a consequence a belief update function is needed. Given a system where agents I and J are performing actions on the environment, the belief update function for

Interactive Partially Observable Markov Decision Processes is as follows:

$$
\begin{aligned}
b_i^t(is^t) = \beta & \sum_{is^{t-1}:\widehat{m}_j^{t-1}=\widehat{\theta}_j^t} b_i^{t-1}(is^{t-1}) \\
\times & \sum_{a^{t-1}\epsilon\{a_i^{t-1}\times A_j\}} T_i(s^{t-1}, a^{t-1}s^t)O_i(s^t, a^{t-1}, o_i^t)P(a_j^{t-1} \mid m_j^{t-1}) \\
\times & \sum_{o_j^t\epsilon\Omega_j} O_j(s^t, a^{t-1}, o_j^t)\tau_{m_j^t}(h_j^{t-1}, a_j^{t-1}, o_j^t, h_j^t)
\end{aligned}
\tag{2.12}
$$

We note that two particular components are introduced in 2.12:

- $P(a_j^{t-1} \mid m_j^{t-1})$: since during the belief update of agent I all the possible actions of J are considered, $P(a_j^{t-1} \mid m_j^{t-1})$ indicates the likelihood of each action depending on the model m_j^{t-1} present in the current interactive state is^{t-1}. The action is calculated by following the criterion indicated in 2.2.
- $\tau_{m_j^t}(h_j^{t-1}, a_j^{t-1}, o_j^t, h_j^t)$: the τ indicates the translation from one model m_j^{t-1} to another m_j^t by means of the actions a_j^{t-1} taken and the possible observation received o_j^t. Due to the way a model is defined in 2.8 this results in a mere update in the history $h_j^{t-1} \rightarrow h_j^t$.

2.2.2 Complexity

Acting optimally in an multi-agent POSD is a very hard task which requires a significant amount of resources. Decentralized POMDPs [13] has been proved to be NEXP-complete [1]. Interactive Partially Observable Markov Decision Processes are very highly intractable due to two major issues[5]:

- *Course of dimensionality*: The belief representation is directly proportional to the dimensions of the belief simplex
- *Course of history*: The dimension of the state of all the policies is proportional to the number of possible future beliefs

These problems, however, are typical ofPartially Observable Markov Decision Processes [14] [15] and, due to the fact that the IPOMDP framework shares various characteristics with it(Bayesian belief update and similar value function) they are transferred to Interactive Partially Observable Markov Decision Processes . Moreover, whenever the modeled agent J is a POMDP type, these characteristics become a part of the I's interactive belief state, hence both the nesting level of a Interactive Partially Observable Markov Decision Process and the types of the simulated agents should be considered when calculating the complexity of the framework. Interactive Partially Observable Markov Decision Process worst-case time complexity seems to be double-exponential [17].

2.2.3 Solving I-POMPs

The whole objective of Interactive Partially Observable Markov Decision Processes is to be able to define the optimal action in a non-deterministic multi-agent system by predicting the actions of the other models. With this in mind it is useful to recall the concept of *optimality criterion* $0 \leq \gamma \geq 1$. The discounted reward is defined as $\sum_{t=0}^{\infty} E(\gamma^t r_t)$. As a consequence, the utility function the agent is trying to maximize is

$$
\begin{aligned}
U(\theta_i) &= \max_{a_i \epsilon A_i}\{\sum_{is} ER(is, a_i)b_i(is) + \gamma \sum_{o_i \epsilon \Omega_i} Pr(o_i \mid a_i, b_i)U(\langle SE_{\theta_i}(b_i, a_i, o_i), \widehat{\theta}_i\rangle)\} \\
&= \max_{a_i \epsilon A_i}\{\sum_{is} \sum_{a_j \epsilon A_j} R_i(is, a_i, a_j)P(a_j \mid m_j)b_i(is) \\
&\quad + \gamma \sum_{o_i \epsilon \Omega_i} Pr(o_i \mid a_i, b_i)U(\langle SE_{\theta_i}(b_i, a_i, o_i), \widehat{\theta}_i\rangle)\}
\end{aligned}
\tag{2.13}
$$

Similarly to the Bellman optimality equation, the I-POMDP's value function can be analyzed in its two parts:

- $\sum_{is} ER(is, a_i)b_i(is)$: This is the part relative to the immediate reward for performing an action a_i in the current interactive belief state b_i
- $\gamma \sum_{o_i \epsilon \Omega_i} Pr(o_i \mid a_i, b_i)U(\langle SE_{\theta_i}(b_i, a_i, o_i), \widehat{\theta}_i\rangle)$: This is the part relative to the future discounted rewards. It takes into account the discounted optimality criterion γ.

It is trivial to note that maximizing 2.13 implies the agent to both branch over all the possible action and observation combinations (due to the future rewards part of the equation) and solving the nested models in the current interactive state. The latter requirement is due to the term $P(a_j \mid m_j)$, which implies being able to solve (or at least estimate) the model m_j in order to be able to calculate a distribution over its possible actions.

Chapter 3

IPOMDPs.jl

The main purpose of this work is to provide the Artificial Intelligence community with an instrument in order to easily define, enhance and test Interactive Partially Observable Markov Decision Processes . The tool proposed is called Julia.IPOMDPs.
It is a framework designed around the user needs to facilitate problem and solver definitions. The project is thought to be an ecosystem of packages interacting together by means of a common interface. As a consequence, the real package *IPOMDPs.jl* only provide such interface, plus some common methods which might be useful to all the future implementations of the framework. The other package provided in this work is a solver: *ReductionSolver.jl* which aims to solve Interactive Partially Observable Markov Decision Processes by folding them in 0-level Partially Observable Markov Decision Processes .

The choice of Julia as the basic programming language is based on its speed and package availability. Julia [2], as explained in 2.1.2, is a programming language designed for high performance. The characteristic that implements multiple dispatch paradigm allows all the future packages of Julia.IPOMDPs framework environment to be developed even quickly due to the simplicity of extension.

The package *IPOMDPs.jl* is the core package of the Julia.IPOMDPs framework. It contains the declarations to be extended in order to define an Interactive Partially Observable Markov Decision Process agent. The agent lifetime is ideally divided into three different phases:

- the *Definition* phase requires the user to define the agent function
- the *Initialization* phase takes includes all those actions needed in order to initialize the program. In this action the initial belief is created and, in the case of online solvers being used, policies are calculated. In this phase the actions performed need to prepare the agent to the next phase:
- the final phase is the *Usage* phase. The agent is prepared and can interact with the environment he is posed in. In order to interact with the agent, an interface is proposed, although it is useful to implement a simulator to automatize the usage process. However, even if in order to test the framework a small simulator has been developed, creating a simulator is not part of this work and it is left to future development.

3.1 Definition

The definition phase is the only phase the user needs to take care of. It is needed in order to define all the logic of the problem. Since Interactive Partially Observable Markov Decision Process is a complex framework, the definition phase must be taken seriously and requires the definition of multiple traits of the agent and the problem structure.
In order to simplify the definition logic and process, this phase has been divided into five major sets:

- *Agent definition* is the part capable of describing the relations among the agents (who is emulating who)
- *Frame definition* is the part used in order to define the proper logic of each agent.
- *Model definition* is the part used in order how a certain model reacts
- *Problem structure* is the part used in order to determine the hierarchy between frames
- *Initial state* is the part used in order to describe the initial state of the problem

3.1.1 Agent

This part is responsible for defining the characteristics of each agent. The agent is characterized by a specific Type, which needs to be declared by taking advantage of the Julia type inheritance. Each agent is different and possesses a determined set of actions and observations:

- `IPOMDPs.agent_actions`
- `IPOMDPs.agent_observations`

Agent actions A_i must be superset of all the actions the models frames $\widehat{m}_{i,n} \epsilon M_i$of and agent I.

$$A_i = \bigcup_{x=1}^{n} A_{\widehat{m}_{i,x}}$$

The same concept needs to be applied to the agent observations

$$\Omega_i = \bigcup_{x=1}^{n} \Omega_{\widehat{m}_{i,x}}$$

3.1.2 Frame

Following the I-POMDP frame definition in 2.4, the frame is the part of the Interactive Partially Observable Markov Decision Process which contains all the information regarding the world transition, the observed behavior, the reward signal from the environment and the optimality criterion for the calculation of the discounted reward. As a result, it comes the need to define all the parts which are themselves part of the IPOMDP framework:

- `IPOMDPs.states`: the set of physical states of the problem. This corresponds to the set S.
- `IPOMDPs.actions`: the joint actions of all agents present in the environment. This corresponds to the set A.
- `IPOMDPs.observations`: the set of observations the current agent can perceive. This corresponds to the set Ω_i.
- `IPOMDPs.transition`: the transition function. This corresponds to T_i.
- `IPOMDPs.observation`: the observation function. This corresponds to O_i.
- `IPOMDPs.reward`: the reward function. This corresponds to R_i.

- `IPOMDPs.discount`: the optimality criteria. It is expressed as a float number in order to limit the depth of the discover tree in the solver used. This corresponds to γ.

It is trivial to note that the states S of the problem must be common among all the agents participating in the game. The user is provided with the ability to define only the physical states of the problem, due to the fact that the interactive states (which are part of the I-POMDP framework as IS_i) are automatically generated.

3.1.3 Problem

The problem phase aims to define the structure of the problem itself. This phase is needed in order to provide a link between the frames of Interactive Partially Observable Markov Decision Processes (and eventually Partially Observable Markov Decision Processes) to the agent they refer to. In the case of I-POMDP frames, this phase is fundamental to indicate which other agent types are to be considered in the generation of the interactive state.

- `IPOMDPs.agent`: specifies which agent the current frame refers to
- `IPOMDPs.emulated_frames`: specifies the frames which are part of the interactive state space of the current I-POMDP frame

Summarizing, this phase connects and specifies the relationship between the frames previously defined.

3.1.4 Initial state

The initial state section defines all the methods necessary in order to initialize the belief of a certain model. In order to increase the expressibility of the framework, the user is required to provide a distribution over both the possible physical states of the world and the frames which will be part of the interactive state set.

- `IPOMDPs.initialstate_distribution`: describes the initial belief distribution regarding the physical states of the environment
- `IPOMDPs.intialframe_distribution`: describes the initial belief distribution regarding the frames emulated by the current I-POMDP frame.

These functions are fundamental in order to be able to determine the initial interactive belief state of the Interactive Partially Observable Markov Decision Process $b_i^{t=0}$.

3.1.5 Model

The *Model* is the operative entity of Julia.IPOMDPs. it is a generalized object containing only a history and a frame. It comes from the formal definition of

$$m = \langle h, f \rangle$$

It needs to be capable of providing a common interface for the program to access the logic of the inner frame. The interface will be referred to as *model interface* and is composed by:

- `IPOMDPs.Model`: Creates the model starting from a defined frame.
- `IPOMDPs.action`: Describes the next optimal action. This function contains the agent function $f(h) \rightarrow A$ described in 2.2.
- `IPOMDPs.actionP`: The probability that a model takes a specific action. It provides $P(a \mid m)$.

- `IPOMDPs.tau`: Updates the history of the model in order to make it keep track of its simulated observations. This is the τ function described in 2.12.
- `IPOMDPs.model_observation`: The way a specific model receives the observation from the environment. This corresponds to O described in 2.2.

The model interface is capable of providing all the functions needed to implement the classical agent life-cycle (Think-Act-Observe). It is important to note that the interface is general enough to make possible to define possibly infinite types of models. It is later shown how to create a model for a Partially Observable Markov Decision Process frame. However, this is not the only type of model we can create. In order to expand the framework with the ability to communicate with different agent types, the user needs to implement the model functions defined above.

In a fictitious case where we want to implement a dummy frame $\widehat{m}_{i,1}$ of the agent I which randomly acts over its possible actions $A_{\widehat{m}_{i,1}} = \{OL, OR\}$ and receive no informations $\Omega_{\widehat{m}_{i,1}} = \{\}$, the relative model would be defined as:

- `IPOMDPs.Model(`$\widehat{m}_{i,1}$`)` $\rightarrow \{nil, \widehat{m}_{i,1}\}$
- `IPOMDPs.action(`$m_{i,1}$`)`$\rightarrow a_i \epsilon A_{\widehat{m}_{i,1}}$
- `IPOMDPs.actionP(`$m_{i,1}$`,`a_i`)`$\rightarrow 0.5$
- `IPOMDPs.tau(`$m_{i,1}$`)`$\rightarrow m_{i,1}$
- `IPOMDPs.model_observation(`$\widehat{m}_{i,1}$`,` nil`)`$\rightarrow 1$

3.2 Initialization

This phase is completely automated and does not need any user interaction. In the Initialization phase of the program, the interactive state set is created, the initial belief over the states is extracted and the eventual policy is calculated. The phase is ideally contained in the previously defined `IPOMDPs.Model`. Due to the way the framework is constructed (3.1.5), the core concept of Julia.POMDPs is the model object. As a consequence, the problem itself is treated as a model object which, by implementing the model interface, is capable to communicate with the environment. In order to create the model of a certain frame, the steps to be taken are:

- *Interactive state set definition*: In this phase, all the interactive states are defined by following the logic defined in 3.1.4. However, in order to be able to perform such operation, all the models of the frames emulated by an I-POMDP need to be defined. This creates a cascade effect whereby calling `IPOMDPs.Model` on the frame of the agent situated at the top of the agent hierarchy, all the models undergo the initialization phase. At the conclusion of this phase, IS_i is formed. It is important to note that, formally, IS_i is an infinite set as defined in 2.6.

  $$IS = S \times M_j \times \cdots \times M_n$$

  However, the set constructed in this phase is a subset of those interactive states which are actually considered by the agent.
- *Initial belief creation*: Once all the interactive states are formed, the initial belief is defined. The process takes advantage of the two previously defined `IPOMDPs.initialstate_distribution` and `IPOMDPs.initialframe_distribution` functions in order to perform a cartesian product and generate the interactive states'probabilities:

  $$P(is) = P(s) \prod_{x \epsilon \{J,\ldots,N\}} P(m_x)$$

In the Interactive Partially Observable Markov Decision Process definition 2.7 the belief is a distribution over all the interactive states IS_i. However, due to the fact that IS_i is infinite, $b_i(IS_i)$ is represented as a discrete distribution over only those states whose probability is different than 0.

- *Policy calculation* is the last operation to perform during the initialization phase. This operation strongly depends on the type of solver used along with POMDPs.jl. In case an off-line solver is used, this phase has the final objective of creating a policy. In the case an online solver is used (as the case of *ReductionSolver.jl*), this phase performs the basic operations in order to set-up the solver, without producing any policy.

Once the model of the frame corresponding to the agent at the top of the agent hierarchy is calculated, the program is ready for the next and final phase.

3.3 Usage

During the usage phase, the program takes advantage of the model interface in order to communicate with the environment. In particular, the functions used by the environment will be:

- `IPOMDPs.action`: represents the acting phase of the agent life cycle. The agent (the model of the problem) calculates the best action depending on its current interactive belief state and returns it to the environment.
- `IPOMDPs.tau`: represents the observation phase of the agent life cycle. It is used in order to provide the model with the observation resulting from the combination of actions of all the agents acting in the environment. `IPOMDPs.tau` is used in order to update the model's interactive belief.

3.3.1 Belief update

In order to keep track of the state of the environment, an Interactive Partially Observable Markov Decision Process keeps track of its interactive belief over physical states and other agents models (2). The Interactive Partially Observable Markov Decision Process update function defined in 2.12, however, is capable to only keep track of the actions and possible observations of two agent I and J. In orde rto be able to really implement I-POMDPs in a true multi-agent environment with N agents, it is needed to expand the original update function:

$$
\begin{aligned}
b_i^t(is^t) = \beta & \sum_{is^{t-1}:\widehat{m}^{t-1}=\widehat{m}^t \forall \widehat{m}} b_i^{t-1}(is^{t-1}) \sum_{a^{t-1}\epsilon\{a_i^{t-1}\times A_j \times \cdots \times A_n\}} T_i(s^{t-1}, a^{t-1} s^t) \\
& \times O_i(s^t, a^{t-1}, o_i^t) \prod_{x\epsilon\{J,\ldots,N\}} P(a_x^{t-1} \mid m_x^{t-1}) \sum_{o_x^t \epsilon \Omega_x} O_x(s^t, a^{t-1}, o_x^t) \\
& \times \tau_{m_x^t}(b_x^{t-1}, a_x^{t-1}, o_x^t, b_x^t)
\end{aligned}
\tag{3.1}
$$

The expansion is minimal but powerful enough to be able to now include the possibility to provide the agent I with infinite constructs over an infinite amount of agents.

However, this function is unpractical to deal with due to the fact that it is based on a summation over IS_i which has infinite cardinality. In order to deal with this problem it turns very useful the approximation adopted in section 3.2, which is to consider only those is whose probability $P(is) \neq 0$. In order to maintain this approximation it is useful to divide the I-POMDP update function into two separate parts:

- *IS expansion*: Expands the set IS. In this first phase, all the possible combinations actions and observations of all the agents in the environment are calculated. They are then passed as a parameter to `IPOMDPs.tau` for all the models in every interactive state. This allows expanding the set of IS to all those interactive states reachable by any combination of actions and observations. The expansion is calculated by considering all the combinations of
  - $is^{t-1} \epsilon IS^{t-1}$
  - $s^t \epsilon S$
  - $a^{t-1} \epsilon \{a_i^{t-1} \times A_j \times \cdots \times A_n\}$
  - $o^t \epsilon \{\Omega_j \times \cdots \times \Omega_n\}$

  Each generated is^t will be defined by

  $$is^t = [s^t, \langle SE(h_j^{t-1}, a_j^{t-1}, o_j^{t-1}), \widehat{m}_j^{t-1} \rangle, \ldots, \langle SE(h_n^{t-1}, a_n^{t-1}, o_n^{t-1}), \widehat{m}_n^{t-1} \rangle] \tag{3.2}$$

  Together with the IS expansion the new probability for each interactive state is calculated:

  $$P(is^t) = b_i^{t-1} T_i(s^{t-1}, a^{t-1}, s^t) O_i(s^t, a^{t-1}, o_i^t) \prod_{x \epsilon J, \ldots, N} P(a_x^{t-1} \mid m_x^{t-1}) O_x(s^t, a^{t-1}, o_x^{t-1}) \tag{3.3}$$

- After the expansion phase it is useful to perform the *duplicate removal* phase. This is a trivial phase but very important. Due to the fact that Bayesian belief update is not bijective [8], there is the possibility to have duplicated is. As a consequence it is useful to aggregate them and sum their probabilities:

  $$P(is^t) = \sum_{is'^{,t} : is'^{,t} = is^t} P(is'^{,t}) \tag{3.4}$$

3.3.2 Action selection

The model object is constructed in such a way that all the information needed in order to calculate the optimal action are already included in the model itself. The next action is obtained by means of the function `IPOMDPs.action(`$m_{i,x}$`)`. The way the model returns the action depends strictly on the model implementation. It will be shown in sections 3.4.1 and 3.4.2 how the action is obtained in the case of a Partially Observable Markov Decision Process and Interactive Partially Observable Markov Decision Process .

3.4 IPOMDOToolbox.jl

In order to make feasible for the first user to use the framework, it is needed to create some initial structures and provides some model and solver implementations. The package *IPOMDPToolbox.jl* aims to provide such basic requirements.

In particular, two model implementations and a solver have been developed to allow future users to work with the framework without needing to define their own solvers:

- `IPOMDOToolbox.ipomdpModel`: is the definition of the model relative to a Interactive Partially Observable Markov Decision Process frame.
- `IPOMDOToolbox.pomdpModel`: is the definition of the model relative to a Partially Observable Markov Decision Process frame.

3.4.1 IPOMDOToolbox.pomdpModel

`IPOMDOToolbox.pomdpModel` is defined in order to allow the user to define Partially Observable Markov Decision Processes in the Julia.IPOMDPs framework. The model is constructed in order to provide an interface with the more famous and structured *Julia.POMDPs* framework. Julia.POMDPs, although it is a relatively new framework, already allows the user to define Partially Observable Markov Decision Processes in a very powerful way and provides an extremely varied array of solvers, benchmark suites and tools to allow users to define POMDPs. In order to take advantage of the expressive power of such framework, `pomdpModel` is designed to act as a wrapper, providing a link between the Julia.IPOMDPs model interface and Julia.POMDPs framework.

`IPOMDPToolbox.pomdpModel` accepts as a frame any Partially Observable Markov Decision Process defined by means of Julia.POMDPs. The most important thing to consider in designing `pomdpModel` is the choice of the solver for the POMDP frame. Two of the most performing solver available at the moment this work has been produced are:

- *SARSOP* [11] is an off-line solver which, during belief exploration, explores only those states reachable by an optimal sequence of actions
- *AR-DESPOT* [20] is an online solver which uses heuristics in order to estimate the value of the policy during the forward search phase.

They are both compatible with Julia.POMDPs interface and hence are considered as candidates for `IPOMDPToolbox.pomdpModel` inner solver. Due to the intrinsic difference between online and off-line solver explained in section 2.1.1, it comes naturally to choose SARSOP as default solver. It is able to compute a policy during the initialization phase and, as a consequence, moves all the complexity to this initial phase, instead of the usage one. Having an already calculated policy means to be able to act only by querying it. This saves a consistent amount of time when $P(a|m)$ needs to be calculated by calling `IPOMDPs.actionP` on the current model.

The model interface is hence implemented as follows:

- `IPOMDPs.Model`: perform the model *Initialization* phase. The belief generated is of type *DiscreteBelief*, which is the one required by SARSOP solver. A SARSOP solver is instantiated and a belief updater is then created and stored in the `pomdpModel` object for later utilization. By providing all the needed structures for `SARSOP.solve` to function, the policy is calculated and stored in the `pomdpModel` object too.
- `IPOMDPs.action`: due to the fact that SARSOP is an off-line solver, the `IPOMDPs.action` method only acts as a wrapper to the `SARSOP.action` function. It passes the computed policy and the current belief stored in the model structure and returns the

selected action to the environment.

- `IPOMDPs.actionP`: Determine the probability of performing a determined action. In order to provide a reliable statistic, the policy is queried 100 times. Thanks to the use of an off-line solver, this procedure is significantly sped up.
- `IPOMDPs.tau`: This is a wrapper for the `SARSOP.update` belief update function. It turns out that the latter is a wrapper too for the more nested `DiscreteBelief.update` function. The used function takes the belief updater and the belief object as a parameter and returns the updated belief. Due to the implementation of how Julia treats the object in memory, it is not possible to update the current model object, but it is required to create a new one. All the references to the static objects in the model (updater, policy, and frame) are passed to the new object along with the updated belief.

3.4.2 IPOMDOToolbox.ipomdpModel

`IPOMDPToolbox.ipomdpModel` is defined in order to allow the user to interact with frames defined by means of IPOMDPs.jl. `ipomdpModel` is, like `pomdpModel` a wrapper to the Interactive Partially Observable Markov Decision Process solver *ReductionSolver.jl* defined in 4.

`IPOMDPToolbox.ipomdpModel` accepts as a frame any Interactive Partially Observable Markov Decision Process defined by means of Julia.IPOMDPs. Due to the fact that at the moment this work has been produced, Julia.IPOMDPs is still being developed, the only choice available is *ReductionSolver.jl*. `ipomdpModel` extends the model interface in the following way:

- `IPOMDPs.Model`: perform the model *Initialization* phase. The obtained belief is of type *DiscreteInteractiveBelief*, which is required by the ReductioSolver solver. The belief updater of type *DiscreteinteractiveUpdater* is defined and stored in the model memory. Due to the fact that the model is using an online solver, no policy is currently calculated. However, a policy object is still generated. This is due to the fact that, due to the solver implementation, the policy is used as a storage for all those elements needed in order to speed up the computation of the model's next action.
- `IPOMDPs.action`: Defines a wrapper method to use `ReductionSolver.action`. While the interface is very similar to the one used by `pomdpModel` to interact with SARSOP solver, the fact that ReductionSolver is an on-line solver profoundly influences the complexity of IPOMDPs.action.
- `IPOMDPs.actionP`: Determine the probability of performing a determined action. In order to provide a reliable statistic, the policy is queried 100 times. Executing `IPOMDPs.actionP` many times by using an on-line solver can prove to be very time-consuming.
- `IPOMDPs.tau`: Similarly to pomdpModel, this acts as a wrapper to the `ReductionSolver.update` function which itself acts as a wrapper for `DiscreteInteractiveBelief.update` which takes as a parameter the belief, updater, observation, and action to calculate the new belief. Like it was described in 3.4.1, the used function takes the belief updater and the belief object as a parameter and returns the updated belief. Due to the implementation of how Julia treats the object in memory, it is not possible to update the current model object, but it is required to create a new one. All the references to the static objects in the model (updater, policy, and frame) are passed to the new object along with the updated belief.

Chapter 4

Solving IPOMDPs

During the past years, a series of solving techniques for solving Interactive Partially Observable Markov Decision Processes have been described and implemented. They are summarized in section 2.2.3. In this work, we try to solve an I-POMDP by reducing it to a POMDP.

In order to explain the methodology it is useful to introduce the concept of strategy level in I-POMDPs. A Interactive Partially Observable Markov Decision Process of agent I with strategy level l is defined as

$$\text{I-POMDP}_{i,l} = \langle IS_{i,l}, A, T_i, O_i, \Omega_i, R_i, OC_i \rangle \tag{4.1}$$

where

$$IS_{i,l} = S \times M_{j,l'<l} \times \cdots \times M_{n,l'<l} \tag{4.2}$$

Is the set of interactive states relative to an I-POMDP of complexity l. The strategy level indicates the level of nesting of the model, meaning the depth of the modeling process. We start with the lower levels $l = 0$, which are POMDPs and all those model types which do not include other agents' belief and frames in their belief space. A depth on 0 means that there is no concept of other agents' models in an agent's belief.

As shown in 4.2 we can see that in the case of an I-POMDP with strategy level $l = 0$, it can be expressed as a POMDP. An agent including 0-*level* (*I-POMDP*$_0$) models in its belief space is said 1-*level* (*I-POMDP*$_1$). Accordingly, we can define a *I-POMDP*$_l$ the one of an agent including *I-POMDP*$_{l-1}$ models in its belief space. Note that we still use the term *I-POMDP*$_l$ for agents including both *I-POMDP*$_{l-1}$ and *I-POMDP*$_{l-x}$ models in its belief space.

Due to this similarity to Partially Observable Markov Decision Process , we can intuitively convert an *I-POMDP*$_n$ to a *I-POMDP*$_0$.

4.1 IPOMDP to POMDP reduction

In order to reduce a Interactive Partially Observable Markov Decision Process to a POMDP, we need to perform a comparison between the I-POMDP definition in 2.2 and the POMDP definition in 2.1. The reduction is performed by comparing each element of the two frameworks and providing a formula capable to link the two definitions which are reported in Appendix A.1:

- *State*: I-POMDP concept of state defined in 2.6 is more complex than POMDP's state definition. However $S \subseteq IS_i$ and, in order to reduce the model, it is possible to marginalize S .
- *Belief*: I-POMDP interactive belief defined in 2.7 might be marginalized to a POMDP belief by marginalizing the models of the agents present in IS_i

$$b_i^t(s) = \sum_{is'^{,t}:s'^{,t}=s^t} b_i^t(is'^{,t}) \tag{4.3}$$

- *Actions*: There is a one-to-one correspondence between the I-POMDP agent actions A_i and the POMDP's action set A. By reminding that the agent actions A_i are the superset of all the models of I's action sets, again $A \subseteq A_i$ and, at the moment we are reducing a specific I-POMDP frame to a POMDP

$$A = A_{\widehat{m}_i}$$

- *Observations*: They can be easily converted, due to the fact that

$$\Omega = \Omega_i$$

- *Transition function*: The transition function is very different in POMDPs and I-POMDPs. The latter model considers the actions of all the agents in the environments in order to predict the change in the physical state, while the former do not include the presence of other players in the game. In order to provide a mapping between the I-POMDP transition and the POMDP transition functions, it is needed to incorporate the actions of all the agents other than I as noise in the transition. This can be formulated as:

$$\begin{aligned} T(s^{t-1}, a_i^{t-1}, s^t \mid b_i) = & \sum_{a_j^{t-1} \epsilon \{A_j \times \cdots \times A_n\}} P(s^t \mid s^{t-1}, a_i^{t-1}, a^{t-1}, b_i^{t-1}) \\ & \times \prod_{x \epsilon \{J,\ldots,N\}} \sum_{m_x \epsilon M_x} P(a_x \mid m_x) P(m_x \mid b_i) \end{aligned} \tag{4.4}$$

  Where the other agents are marginalized by calculating the probability of their actions and of their models.
- *Observation function*: The same concept as above is applied to the observation function. The other agents are included as environmental noise and the received observations probabilities reflect the probabilities the other agents to perform certain actions:

$$\begin{aligned} O(s^t, a_i^{t-1}, o_i^t \mid b_i) = & \sum_{a_j^{t-1} \epsilon \{A_j \times \cdots \times A_n\}} P(o_i^t \mid s^t, a_i^{t-1}, a^{t-1}, b_i^{t-1}) \\ & \times \prod_{x \epsilon \{J,\ldots,N\}} \sum_{m_x \epsilon M_x} P(a_x \mid m_x) P(m_x \mid b_i) \end{aligned} \tag{4.5}$$

- *Reward function*: The third function where the main difference between I-POMDP and POMDP definition is the inclusion of the other agents' actions in the reward function. The approach is the same taken for the transition and observation functions:

$$R(s, a_i \mid b_i) = \sum_{a \epsilon \{A_j \times \cdots \times A_n\}} R_i(s, a_i, a) \prod_{x \epsilon \{J,\ldots,N\}} \sum_{m_x \epsilon M_x} P(a_x \mid m_x) P(m_x \mid b_i) \tag{4.6}$$

- *Optimality criterion*: It is independent on the framework implementation. In fact, it is straightforward to define:

$$\gamma_{pomdp} = \gamma_{ipomdp}$$

- *Belief update*: While I-POMDP and POMDP belief update functions look similar for they require the same parameters $(b_i^{t-1}, a_i^{t-1}, o_i^t)$ their inner behavior is profoundly different. Updating an Interactive Partially Observable Markov Decision Process requires to consider all the possible observations and actions of all the other models, other than estimating an updated version of them. This is completely absent in the POMDP updated function. As a consequence, the update function is not reducible. This leads to problems in terms of precision of a solver implementing such a technique. It has been shown in 4.4, 4.5, 4.6, and 4.3 that the generated POMDP elements are directly proportional to $P(a_x \mid m_x)P(m_x \mid b_i)$. However, these terms might change each time the agent's belief needs to be changed. This leads to generated POMDP which is consistent with the I-POMDP only for the current time t. However, solution algorithms to Partially Observable Markov Decision Processes do not consider the possibility for the agent model to change at each time step. This inconvenience reduces the precision of any solver which implements the I-POMDP to POMDP reduction technique.
- *Utility function*: Utility for a I-POMDP is described in 2.13. Such function is composed by several parts:
  - The *immediate reward* part for both IPOMDP and POMDP utility can be reduced by means of equation 4.6.
  - The part related to the *future reward* seems already identical between POMDP and I-POMDP utility function definitions.

  However, the future reward part includes the belief update function. Due to the fact that the belief update is not consistent, the utility function should be considered coherent only for the current time step t and hence only for its immediate reward section.

4.2 Julia.IPOMDPs

In order to provide Julia.IPOMDPs with the ability to define and extend solvers it is needed to design an interface each solver should implement. Such an interface is referred to as the *solver interface*. It is strictly related to the solver definition in Julia.POMDPs [6] and is defined as:

- `IPOMDPs.updater`: Provides the user with the belief update the solver is designed to use.
- `IPOMDPs.initialize_belief`: Provides the initial belief $b_i^{t=0}$ state for the problem. This function acts as a wrapper for the real belief initialization function.
- `IPOMDPs.update`: Updates the belief by using the logic provided by the belief updater. This method is often used as a wrapper for the real belief update function.
- `IPOMDPs.solve`: provides the user with a policy. This method can be profoundly different in the online and off-line solver. In the former type, this method is very light and it is just used in order to create a policy object which will be propagated for all the agent's lifetime. On-line solvers do not need to carry policies over the time since they generate the current policy at time t. However, a policy object could be extremely useful to carry information used in order to significantly speed up the

computation. In the case of Off-line solvers, this method includes a major part of the complexity of solving Interactive Partially Observable Markov Decision Processes .

- `IPOMDPs.action`: is the complementary method of `IPOMDPs.solve`. It is used in order to determine the optimal action for a certain belief state given the policy object. In the case of online solvers, this method includes the majority of the complexity of the problem. The solver, in fact, needs to partially solve the problem and determine the current optimal action. For off-line solvers, instead, `IPOMDPs.action` is used in order to query the policy and determine which is the optimal strategy.

4.3 ReductionSolver.jl

ReductionSolver.jl is the module of Julia.IPOMDPs developed in order to provide the user with the means to solve Interactive Partially Observable Markov Decision Processes . It is an online solver, meaning that the optimal action is calculated depending only on the current belief state and do not include the presence of any policy. In order to design ReductionSolver, it has been followed an approach significantly different from the one taken by all the solvers present in the I-POMDP solver ecosystem. While most of the solvers rely on iteration methods such as value iteration or policy iteration in order to calculate the value of a policy and a specific interactive belief, reduction solver tries to solve the maximization problem by reducing the I-POMDP to a POMDP and relying on the existent ecosystem of solvers already available to solve Partially Observable Markov Decision Processes .

In order to implement the functions described in section 4.2 and hence provide all the necessary functionalities, it is first needed to define elements such as belief type, belief updater, policy type, and solver object:

- *DiscreteInteractiveBelief*: The interactive belief object is the implementation of a probability mass function on the discrete variable IS_i
- *DiscreteInteractiveUpdater*: It is the object responsible of the behavior of the IPOMDP agent belief update and initialization. It implements the belief initialization function by extending the solver interface function `IPOMDPs.initialize_belief` and providing the first two operations described in 3.2: *Interactive state set definition* and *Initial belief creation*.
- *ReductionPolicy*: The policy object is defined in order to maintain data produced during successive runs of the solving algorithm. It is passed along with the belief to`IPOMDPs.action`. The implementation contains a table of all those POMDPs which have already been converted and solved. This element is necessary at each phase present in the *Usage* phase described in section 3.3.
- *ReductionSolver*: the solver object is intended as a container for all the possible settings needed for the solver. It is passed as an argument to the `IPOMDPs.solve` function and takes part in the policy generation process.

4.3.1 IPOMDP reduction

ReductionSolver takes advantage of the existent ecosystem of POMDP solvers by reducing I-POMDPs to Partially Observable Markov Decision Processes . The actual conversion is performed by defining a special POMDP type called `gPOMDP`. `gPOMDP` is defined by relying on Julia.POMDPs interface. All the POMDP elements reflect the relative equations defined in 4.1, which are converted in order to make them easily computable. After the conversion process, the general POMDP is solved by means fo the SARSOP off-line solver. In order to provide a more accurate solution, the problem is solved by using as initial state distribution the current reduced belief. Once the `gPOMDP` object is solved and a policy created the optimal action is generated.

4.3.2 Curse of history reduction

Due to the fact that `ReductionSolver` is an online solver, a real policy does not exist. However, the implementation structure of `IPOMDPs.action` requires a policy object to be passed as an argument. In order to speed up the process of calculating the optimal action, the policy object can be used as a storage for solved `gPOMDP` problems. In fact, `gPOMDPs` are not unique. They could be equal to other already solved if the *I-POMDP* models used in order to generate them are identical both in the frame and the belief over interactive states. As a consequence it is possible, after defining a suitable comparing function, to define a table to store solved `gPOMDPs` to avoid repeated calculations. This tweak allows to significantly reduce the time needed for calculating the optimal action. In fact, without it, solving 2-level I-POMDPs would be nearly impossible. As a consequence, the impact of the curse of history, while it is not completely avoided, it is significantly mitigated by storing the known solution to already visited problems.

Chapter 5

Tests

In order to correctly test the solver and to show the possible applications, it could be useful to define some hypothetical games and compare the performances with other existent solvers.

5.1 Test environment

Tiger game (Appendix B) is a theoretical game proposed in [3] and is reported in where an agent is posed in front of two doors. Behind a door stands a tiger and behind the other one stands a reward. If the agent opens the wrong door receives a large penalty. The agent uses the infinite horizon with discounted rewards optimality criterion in order to calculate the best action.

The problem is particularly suited for Partially Observable Markov Decision Process because it includes an information gathering phase, implemented through the possibility of the agent listening. In this section, it will be derived a variant of such game which is used as a reference game for testing Julia.IPOMDPs.

The multi-agent tiger game (Appendix C) described in this paper is a modification of the multi-agent game described in [7], which is itself derived from the single-agent tiger game. The environment includes two agents I and J which are in front of two doors. Only one door hides the reward, while the other hides a tiger. As a consequence, we can define

$$S = \{TL, TR\}$$

Each agent can perform one of these three actions:

$$A_i = A_j = \{OL, L, OR\}$$

and, depending on the state the action is taken, the agent is rewarded accordingly. Similarly to the original tiger game, the agent can receive observations but in this case, they are expanded. It has been included the possibility to hear creeks generated by J's actions. However, these observations are noisy.

$$\Omega_i = \{GLCL, GLCR, GLS, GRCL, GRCR, GRS\}$$

The environment is a non-deterministic environment which can generate transitions following C.1, which are reported below:

$\langle a_i, a_j \rangle$	**State**	**TL**	**TR**
$\langle OL, * \rangle$	*	0.5	0.5
$\langle OR, * \rangle$	*	0.5	0.5
$\langle *, OL \rangle$	*	0.5	0.5
$\langle *, OR \rangle$	*	0.5	0.5
$\langle L, L \rangle$	TL	1.0	0.0
$\langle L, L \rangle$	TR	0.0	1.0

Table 5.1: Multi-agent tiger game transition function

5.2 Multi-agent tests

There are some interesting implications of applying Interactive Partially Observable Markov Decision Processes in a multi-agent environment. The Most interesting one is that with a 1-level agent it is possible to define the behavior I has with respect to J. The three versions of the agent have been denominated:

- *Neutral*: Agent I does not pay any interest to J. It emulates the other agent in order to better predict the environment.
- *Cooperative*: Agent I is designed as a friend of J. Is reward function is declared in C.7 and it is created in order to maximize I's reward when also J opens the correct door.
- *Competitive*: Agent I in this case is an enemy of J. Is reward function is declared in C.6 and it is created in order to maximize I's reward when J opens the wrong door

In order to define the behavior of the agents it is sufficient to change their reward function. Here are reported the reward functions described in C.4, C.7 and C.6:

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, * \rangle$	-100.0	10.0
$\langle OR, * \rangle$	10.0	-100.0
$\langle L, * \rangle$	-1.0	-1.0

(a) Neutral agent

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, OL \rangle$	-150.0	15.0
$\langle OL, OR \rangle$	-95.0	-40.0
$\langle OL, L \rangle$	-100.5	9.5
$\langle OR, OL \rangle$	-40.0	-95.0
$\langle OR, OR \rangle$	15.0	-150.0
$\langle OR, L \rangle$	9.5	-100.5
$\langle L, OL \rangle$	-51.0	4.0
$\langle L, OR \rangle$	4.0	-51.0
$\langle L, L \rangle$	-1.5	-1.5

(b) Cooperative agent

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, OL \rangle$	-50.0	5.0
$\langle OL, OR \rangle$	-105.0	60.0
$\langle OL, L \rangle$	-99.5	10.5
$\langle OR, OL \rangle$	60.0	-105.0
$\langle OR, OR \rangle$	5.0	-50.0
$\langle OR, L \rangle$	10.5	-99.5
$\langle L, OL \rangle$	49.0	-6.0
$\langle L, OR \rangle$	-6.0	49.0
$\langle L, L \rangle$	-0.5	-0.5

(c) Competitive agent

Table 5.2: Possible Agent I Reward functions

The agents receive observations from the environment by following:

$\langle a_i, a_j \rangle$	**State**	**GLCL**	**GLCR**	**GLS**	**GRCL**	**GRCR**	**GRS**
$\langle OL, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle L, OL \rangle$	TL	0.765	0.043	0.043	0.135	0.007	0.007
$\langle L, OL \rangle$	TR	0.135	0.007	0.007	0.765	0.043	0.043
$\langle L, OR \rangle$	TL	0.043	0.765	0.043	0.007	0.135	0.007
$\langle L, OR \rangle$	TR	0.007	0.135	0.007	0.043	0.765	0.043
$\langle L, L \rangle$	TL	0.043	0.043	0.765	0.007	0.007	0.135
$\langle L, L \rangle$	TR	0.007	0.007	0.135	0.043	0.043	0.765

Table 5.3: Multi-agent tiger game Agent I observation function

In the case one agent participating in the game is a Partially Observable Markov Decision Process , some adaptations should be performed. Since POMDP is a single-agent framework, it is useless to provide the agent with observations regarding the other agent's actions (meaning *creek* and *silence*). It is considered that agent J simply ignores such information, hereby, as described in table B.3, the world observations will result in:

a	**State**	**GL**	**GR**
OL	*	0.5	0.5
OR	*	0.5	0.5
L	TL	0.85	0.15
L	TR	0.15	0.85

Table 5.4: Multi-agent tiger game simplified observation function

5.2.1 Performances

There are two main important factors to consider when profiling an algorithm: space and time performances. These aspects are particularly crucial in a Interactive Partially Observable Markov Decision Process simulator due to its complexity.

- *Execution time*: It is the time needed in order to perform one agent loop. The two main operations are selecting the action and updating the model.
- *Memory consumption*: It is the amount of memory necessary in order to store all the necessary data structures.
- *Number of distinct interactive states*: It is used as an index on the level of complexity of the problem. The more interactive states are present, the more computation time is expected to be used, due to the fact that more simulations need to be performed.

In order to provide useful statistics, a test over *1000* rounds has been performed. The feature extracted are the mean values on

- *action time*: It is the time needed for the function *IPOMDPs.action* to return the optimal action.

- *update time*: It is the time that *IPOMDPs.update* needs in order to update the model of I given the past action and current observation.
- *memory allocated*: Indicates the number of kB that have been allocated during the runtime of both *IPOMDPs.action* and *IPOMDPs.update*. Note that this value might not be the exact amount of memory used by the program, due to the fact that there are other factors which influence the runtime memory, such as garbage collector and other functions allocations.
- IS *size*: The average number of interactive states present during execution time.

The following data has been recorded during execution and later parsed by means of *matlab*. The precision is indicated as standard deviation.

	action time(s)	**update time**(s)	**memory allocated** (kB)	IS **size**
Neutral	0.8146 ± 0.7745	0.0273 ± 0.0080	10398.4309 ± 1280.0033	9.9860
Cooperative	0.9029 ± 0.7990	0.0296 ± 0.0527	$10935.6964 \pm 13165.7382$	9.9860
Competitive	0.8337 ± 0.7136	0.0272 ± 0.0080	10259.9010 ± 1524.9730	9.9860

Table 5.5: Level 1 Tiger game run times statystics for 1000 runs

In order to analyze the complexity of the problem, it is possible to analyze the number of interactive states generated from the problem during the time.

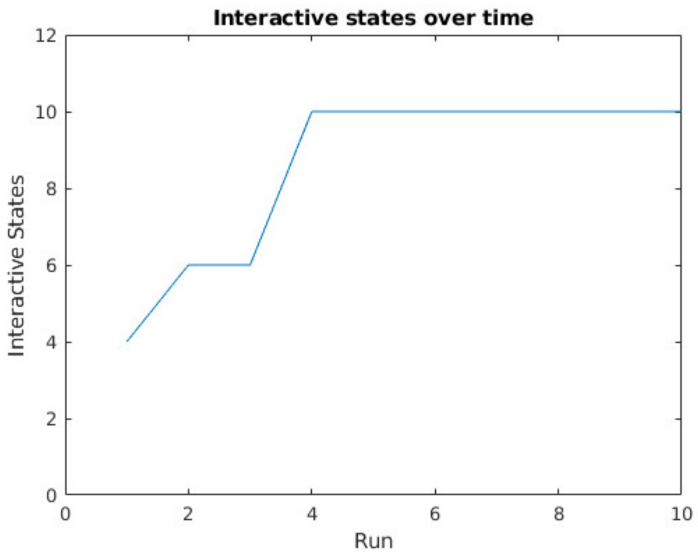

Figure 5.1: Number of Interactive States in each run

It is clear to note in figure 5.1 that the program tends to convert to a fixed amount of interactive states. The interactive states set size is relatively low due to the fact that they depend on the possible combination of J belief states which are shown in 5.6 and the physical states S of the system creating 10 interactive states as expected.

	P(TL)	P(TR)
1	0.5	0.5
2	0.85	0.15
3	0.85	0.15
4	0.97	0.03
5	0.03	0.97

Table 5.6: Possible J beliefs

The various settings of the multi-agent tiger game are useful in order to test the influence the agent I can have on the agent J reward factor. In the second experiment, the various agents are playing in the same environment with the POMDP tiger agent. It is possible to note the influence of the agent I on J's rewards. This concept is useful since it shows the applicability of Interactive Partially Observable Markov Decision Processes to those problems where the influence one agent can have on the other is fundamental, like an assistant.

Example	mean
Cooperative	9.2084
Neutral	7.1288
Competitive	-4.5190

Table 5.7: Average discounted reward for Agent J with $\gamma = 0.95$

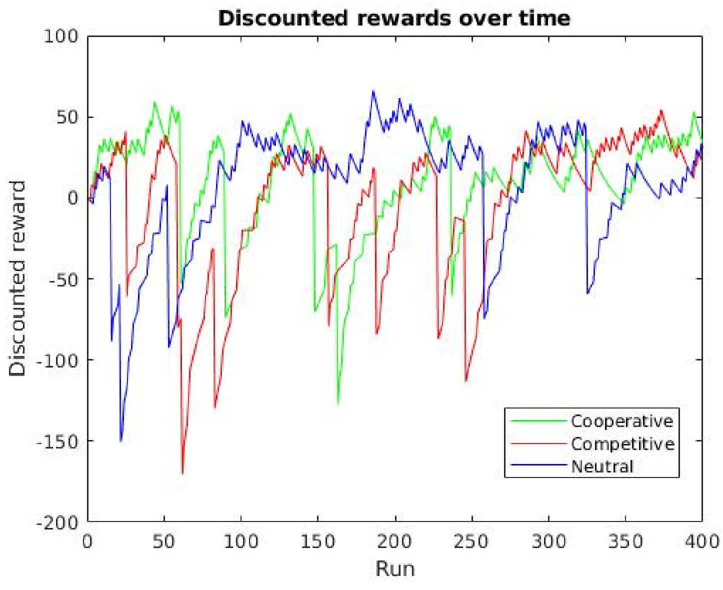

Figure 5.2: Discounted reward for Agent J with $\gamma = 0.95$

5.7 and 5.2 are formed by picking 400 samples from the whole execution dataset. The result shows how the actions of I and its behavior influence the reward of J. However, even if in 5.2 the behaviors are rather noisy, it is possible to recognize the agents by the number of lower peaks of the three functions. The cooperative agent allows J to make less errors than both the neutral and competitive agents.

5.3 2-Level multi-agent tiger game

One of the main strength of the Interactive Partially Observable Markov Decision Process framework is the possibility to define agents on various complexity levels. Julia.IPOMDPs is structured in order to take full advance of this strength and make it as easy as possible for the user to define nested models. In the 2-level multi-agent tiger game both the agents are described as Interactive Partially Observable Markov Decision Processes : while I is a 2-level I-POMDP, J is a 1-level I-POMDP. As a consequence I is emulating J which is itself emulating I (this time as a 0-level POMDP). This example is relatively easy, but it is enough in order to allow us to understand one of the main problems of Interactive Partially Observable Markov Decision Processes : Computational and space complexity as described in 2.2.2. The same data taken for 5.5 is taken. However, no mean value is calculated due to the scarcity of elements.

run	**action time**(s)	**update time**(s)	**memory allocated** (kB)	IS **size**
1	9.2261	8.6383	415,816.0500	4
2	49.1595	40.7981	1,153,249.4400	6
3	125.6628	105.5685	2,890,821.1360	14
4	455.9217	385.2239	11,104,481.3120	28
5	1,701.2110	1,419.9813	42,872,848.4000	116

Table 5.8: Agent I run-time for 2-level I-POMDP

In fact, run times are significantly higher than the ones obtained in the former examples. This is expected due to the increased complexity of the problem. There are now more interactive states to parse and, moreover, it is necessary to recurse more deeply in the model structure. Even if some precautions in order to improve the execution time have been adopted in section 4.3.2, it is interesting to note how the execution and the memory time increase due to the curse of history. As a consequence we can confirm that the explored `gPOMDP` table is useful only in case the program visits again the same belief state (e.g. during `IPOMDPs.actionP`).

5.4 Agent model learning

Another interesting use of the Interactive Partially Observable Markov Decision Process framework is to use its belief update function in order to learn the other agents' model. In this experiment, the task of agent J is still to maximize the multi-agent tiger game reward, but the data we are interested in is its capability of recognizing the agent it is interacting with.

In order to set up the experiment, we need to define the possible agent behaviors J can assume:

- *normal agent*: This agent acts following the rules of the original tiger game.
- *suicidal agent*: This agent acts in the opposite way of the other. It is rewarded whenever it is eaten by the tiger and it gets a strong penalty when it fails to do so.

I starts without any information neither on which agent he is playing with or the tiger location and as a consequence its initial belief is:

	TL	**TR**	ϕ
$\theta_{j,n}$	0.25	0.25	0.5
$\theta_{j,s}$	0.25	0.25	0.5

The aim of the test is to recognize which agent I is playing with. After 250 runs in a simulator where I is playing against the suicidal agent, the precision is extremely high as shown in table 5.3:

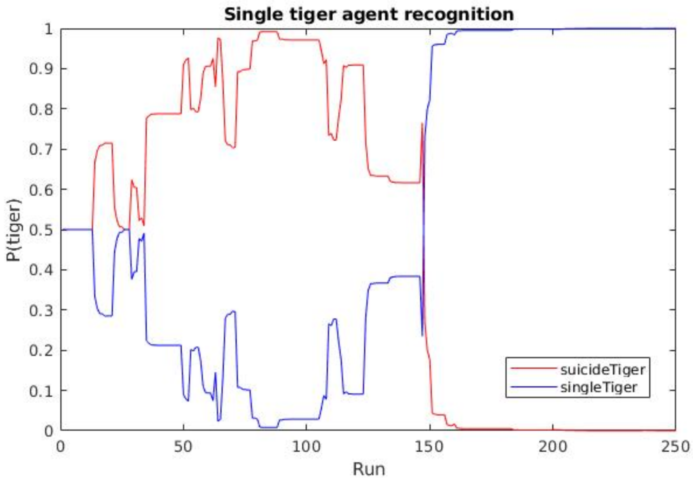

Figure 5.3: Recognition of the normal Tiger POMDP

Learning the other agent's model is very useful because allows the agent to take actions depending on the behavior of the other agents it is interacting with.

5.5 Conclusions

Development of Interactive Partially Observable Markov Decision Processes by means of Julia.IPOMDPs results smooth and straight-forward thanks to its interface architecture. Due to the nature of the thesis work, it has not been possible to control all the parts of developing such a project. There are several improvements which are left for future development and possible research area. It has been shown the procedure used in order to create the framework, but creating a software does not only require the design and implementation phase. One immediate improvement of the work could be by implementing a regression testing suite in order to provide consistent results in the future possible releases. Moreover, Julia.IPOMDPs does not provide a simulator capable of easily handling the agent testing phase. Reduction solver, even if it is affected by the problem explained in 4.1, proves beyond expectations in terms of speed. However, it cannot be considered as an exact I-POMDP solver. Whether or not it is possible to implement a correct reduction respecting the POMDP belief update is left a matter of further research.

Appendix A

Reduction formulas

A.1 Transition reduction formula

$$
\begin{aligned}
T(s^{t-1}, a_i^{t-1}, s^t \mid b_i) &= P(s^t \mid s^{t-1}, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(s^t \mid s^{t-1}, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) P(a_j^{t-1} \mid s^{t-1}, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(s^t \mid s^{t-1}, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \\
&\times \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid s^{t-1}, a_i^{t-1}, b_i^{t-1}, m_j^{t-1}) P(m_j^{t-1} \mid s^{t-1}, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(s^t \mid s^{t-1}, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid m_j^{t-1}) \\
&\times P(s^{t-1}, a_i^{t-1}, b_i^{t-1}) P(m_j^{t-1} \mid b_i^{t-1}) P(s^{t-1}, a_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(s^t \mid s^{t-1}, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \\
&\times \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid m_j^{t-1}) P(m_j^{t-1} \mid b_i^{t-1})
\end{aligned}
\tag{A.1}
$$

A.1.1 Note: a_j^{t-1} indep. $s^{t-1}, a_i^{t-1}, b_i^{t-1}$ given m_j^{t-1}:

It is part of the *IPOMDP* framework definition that the action a_j is determined only by the model m_j.

A.1.2 Note: m_j^{t-1} indep. s^{t-1}, a_i^{t-1} given b_i^{t-1}:

It is part of the *IPOMDP* framework definition that the action a_j is determined only by the model m_j. Moreover, m_j is part of b_i^{t-1}.

A.2 Observation reduction formula

$$
\begin{aligned}
O(s^t, a_i^{t-1}, o_i^t \mid b_i) &= P(o_i^t \mid s^t, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(o_i^t \mid s^t, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) P(a_j^{t-1} \mid s^t, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(o_i^t \mid s^t, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \\
&\times \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid s^t, a_i^{t-1}, b_i^{t-1}, m_j^{t-1}) P(m_j^{t-1} \mid s^t, a_i^{t-1}, b_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(o_i^t \mid s^{t-1}, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid m_j^{t-1}) \\
&\times P(s^t, a_i^{t-1}, b_i^{t-1}) P(m_j^{t-1} \mid b_i^{t-1}) P(s^t, a_i^{t-1}) \\
&= \sum_{a_j^{t-1} \epsilon A_j} P(o_i^t \mid s^t, a_i^{t-1}, a_j^{t-1}, b_i^{t-1}) \\
&\times \sum_{m_j^{t-1} \epsilon M_j} P(a_j^{t-1} \mid m_j^{t-1}) P(m_j^{t-1} \mid b_i^{t-1})
\end{aligned}
\tag{A.2}
$$

A.2.1 Note 4: a_j^{t-1} indep. $s^t, a_i^{t-1}, b_i^{t-1}$ given m_j^{t-1}:

It is part of the *IPOMDP* framework definition that the action a_j is determined only by the model m_j.

A.2.2 Note 5: m_j^{t-1} indep. s^t, a_i^{t-1} given b_i^{t-1}:

It is part of the *IPOMDP* framework definition that the action a_j is determined only by the model m_j. Moreover, m_j is part of b_i^{t-1}.

A.3 Reward reduction formula

$$
\begin{aligned}
&R(s, a_i \mid b_i) : \\
&\qquad = \sum_{a_j \epsilon A_j} R_i(s, a_i, a_j) P(a_j \mid b_i) \\
&\qquad = \sum_{a_j \epsilon A_j} R_i(s, a_i, a_j) \sum_{m_j \epsilon M_j} P(a_j \mid m_j, b_i) P(m_j \mid b_i) \\
&\qquad = \sum_{a_j \epsilon A_j} R_i(s, a_i, a_j) \sum_{m_j \epsilon M_j} P(a_j \mid m_j) P(b_i) P(m_j \mid b_i) \\
&\qquad = \sum_{a_j \epsilon A_j} R_i(s, a_i, a_j) \sum_{m_j \epsilon M_j} P(a_j \mid m_j) P(m_j \mid b_i)
\end{aligned}
\tag{A.3}
$$

A.3.1 Note: a_j indep. b_i given m_j:

It is part of the *IPOMDP* framework definition that the action a_j is determined only by the model m_j. Moreover m_j is part of the belief b_i

Appendix B

POMDP Tiger game definition

B.1 States

$$S = \{TL, TR\}$$

B.2 Actions

$$A = \{L, OL, OR\}$$

B.3 Observations

$$\Omega = \{GL, GR\}$$

B.4 Transition function

a	**State**	**TL**	**TR**
OL	*	0.5	0.5
OR	*	0.5	0.5
L	TL	1.0	0.0
L	TR	0.0	1.0

Table B.1: POMDP Tiger game transition function

B.5 Observation function

a	**State**	**GL**	**GR**
OL	*	0.5	0.5
OR	*	0.5	0.5
L	TL	0.85	0.15
L	TR	0.15	0.85

Table B.2: POMDP Tiger game observation function

B.6 Reward function

a	**TL**	**TR**
OL	-100.0	10.0
OR	10.0	-100.0
L	-1.0	-1.0

Table B.3: Tiger game Agent J Reward function

Appendix C

Multi-agent Tiger game definition

Definition for Agent I multi-agent Tiger game.

Agent J is considered to be defined as in Appendix B

C.1 States

$$S = \{TL, TR\}$$

C.2 Agent actions

$$A_i = \{OL, L, OR\}$$

C.3 Observations

$$\Omega_i = \{GLCL, GLCR, GLS, GRCL, GRCR, GRS\}$$

C.4 Transition function

$\langle a_i, a_j \rangle$	**State**	**TL**	**TR**
$\langle OL, * \rangle$	*	0.5	0.5
$\langle OR, * \rangle$	*	0.5	0.5
$\langle *, OL \rangle$	*	0.5	0.5
$\langle *, OR \rangle$	*	0.5	0.5
$\langle L, L \rangle$	TL	1.0	0.0
$\langle L, L \rangle$	TR	0.0	1.0

Table C.1: Multi-agent tiger game transition function

C.5 Observation functions

$\langle a_i, a_j \rangle$	**State**	**GLCL**	**GLCR**	**GLS**	**GRCL**	**GRCR**	**GRS**
$\langle OL, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle L, OL \rangle$	TL	0.765	0.043	0.043	0.135	0.007	0.007
$\langle L, OL \rangle$	TR	0.135	0.007	0.007	0.765	0.043	0.043
$\langle L, OR \rangle$	TL	0.043	0.765	0.043	0.007	0.135	0.007
$\langle L, OR \rangle$	TR	0.007	0.135	0.007	0.043	0.765	0.043
$\langle L, L \rangle$	TL	0.043	0.043	0.765	0.007	0.007	0.135
$\langle L, L \rangle$	TR	0.007	0.007	0.135	0.043	0.043	0.765

Table C.2: Multi-agent tiger game Agent I observation function

$\langle a_j, a_i \rangle$	**State**	**GLCL**	**GLCR**	**GLS**	**GRCL**	**GRCR**	**GRS**
$\langle OL, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OL, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OL \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, OR \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TL	0.167	0.167	0.167	0.167	0.167	0.167
$\langle OR, L \rangle$	TR	0.167	0.167	0.167	0.167	0.167	0.167
$\langle L, OL \rangle$	TL	0.765	0.043	0.043	0.135	0.007	0.007
$\langle L, OL \rangle$	TR	0.135	0.007	0.007	0.765	0.043	0.043
$\langle L, OR \rangle$	TL	0.043	0.765	0.043	0.007	0.135	0.007
$\langle L, OR \rangle$	TR	0.007	0.135	0.007	0.043	0.765	0.043
$\langle L, L \rangle$	TL	0.043	0.043	0.765	0.007	0.007	0.135
$\langle L, L \rangle$	TR	0.007	0.007	0.135	0.043	0.043	0.765

Table C.3: Multi-agent tiger game Agent J observation function

C.6 Reward function

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, * \rangle$	-100.0	10.0
$\langle OR, * \rangle$	10.0	-100.0
$\langle L, * \rangle$	-1.0	-1.0

Table C.4: Multi-agent tiger game Agent I Reward function

$\langle a_j, a_i \rangle$	**TL**	**TR**
$\langle OL, * \rangle$	-100.0	10.0
$\langle OR, * \rangle$	10.0	-100.0
$\langle L, * \rangle$	-1.0	-1.0

Table C.5: Multi-agent tiger game Agent J Reward function

C.7 Variations

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, OL \rangle$	-50.0	5.0
$\langle OL, OR \rangle$	-105.0	60.0
$\langle OL, L \rangle$	-99.5	10.5
$\langle OR, OL \rangle$	60.0	-105.0
$\langle OR, OR \rangle$	5.0	-50.0
$\langle OR, L \rangle$	10.5	-99.5
$\langle L, OL \rangle$	49.0	-6.0
$\langle L, OR \rangle$	-6.0	49.0
$\langle L, L \rangle$	-0.5	-0.5

Table C.6: Competitive tiger game Agent I Reward function

$\langle a_i, a_j \rangle$	**TL**	**TR**
$\langle OL, OL \rangle$	-150.0	15.0
$\langle OL, OR \rangle$	-95.0	-40.0
$\langle OL, L \rangle$	-100.5	9.5
$\langle OR, OL \rangle$	-40.0	-95.0
$\langle OR, OR \rangle$	15.0	-150.0
$\langle OR, L \rangle$	9.5	-100.5
$\langle L, OL \rangle$	-51.0	4.0
$\langle L, OR \rangle$	4.0	-51.0
$\langle L, L \rangle$	-1.5	-1.5

Table C.7: Cooperative tiger game Agent I Reward function

Bibliography

[1] Bernstein, D. S., Zilberstein, S., and Immerman, N. The complexity of decentralized control of markov decision processes. *Math. Oper. Res. 27* (2000), 819–840.

[2] Bezanson, J., Karpinski, S., Shah, V. B., and Edelman, A. Julia: A fast dynamic language for technical computing. *CoRR abs/1209.5145* (2012).

[3] Cassandra, A. R., Kaelbling, L. P., and Littman, M. L. Acting optimally in partially observable stochastic domains. In *AAAI* (1994).

[4] Doshi, P. Decision making in complex multiagent contexts: A tale of two frameworks. *AI Magazine 33* (2012), 82–95.

[5] Doshi, P., and Gmytrasiewicz, P. J. Monte carlo sampling methods for approximating interactive pomdps. *J. Artif. Intell. Res. 34* (2009), 297–337.

[6] Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., and Kochenderfer, M. J. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research 18*, 26 (2017), 1–5.

[7] Gmytrasiewicz, P. J., and Doshi, P. A framework for sequential planning in multi-agent settings. *J. Artif. Int. Res. 24*, 1 (July 2005), 49–79.

[8] Gmytrasiewicz, P. V. . P. Sampling & updating higher order beliefs, 2010.

[9] Han, Y., and Gmytrasiewicz, P. J. Learning others' intentional models in multi-agent settings using interactive pomdps. In *MAICS* (2017).

[10] Hansen, E. A., Bernstein, D. S., and Zilberstein, S. Dynamic programming for partially observable stochastic games. In *AAAI* (2004).

[11] Kurniawati, H., Hsu, D., and Lee, W. S. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems* (2008).

[12] Ng, B., Meyers, C., Boakye, K., and Nitao, J. J. Towards applying interactive pomdps to real-world adversary modeling. In *IAAI* (2010).

[13] Oliehoek, F. A. Decentralized pomdps.

[14] Pineau, J., Gordon, G. J., and Thrun, S. Anytime point-based approximations for large pomdps. *J. Artif. Intell. Res. 27* (2006), 335–380.

[15] Poupart, P., and Boutilier, C. Vdcbpi: an approximate scalable algorithm for large pomdps. In *NIPS* (2004).

[16] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[17] Seuken, S., and Zilberstein, S. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems 17* (2007), 190–250.

[18] Seymour, R. S., and Peterson, G. L. Responding to sneaky agents in multi-agent domains. In *FLAIRS Conference* (2009).

[19] Shapley, L. S. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America 39 10* (1953), 1095–100.

[20] Somani, A., Ye, N., Hsu, D., and Lee, W. S. Despot: Online pomdp planning with regularization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (USA, 2013), NIPS'13, Curran Associates Inc., pp. 1772–1780.

[21] Sonu, E., and Doshi, P. Generalized and bounded policy iteration for finitely-nested interactive pomdps: scaling up. In *AAMAS* (2012).

[22] Sonu, E., and Doshi, P. Scalable solutions of interactive pomdps using generalized and bounded policy iteration. *Autonomous Agents and Multi-Agent Systems 29*, 3 (May 2015), 455–494.

[23] Spaan, M. Multiagent models for partially observable environments, 2007.

[24] Ye, N., Somani, A., Hsu, D., and Lee, W. S. Despot: Online pomdp planning with regularization. In *NIPS* (2013).