Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

# Best response computation for partially observable stochastic games

Ondřej Kubíček

Supervisor: Mgr. Branislav Bošanský, Ph.D.
Field of study: Cybernetics and Robotics
May 2020

## Acknowledgements

I would like to thank my supervisor Mgr. Branislav Bošanský, Ph.D. for his guidance and expertise during the work on this thesis. I would also like to thank Ing. Karel Horák for his contribution to research on Partially Observable Stochastic Games. Without his research this thesis may not be possible.

## Declaration

## Abstract

Solving Partially Observable Stochastic games (POSG) is computationally hard in general, therefore the research is focused on variants of POSGs. One-Sided POSGs are played by two players, where one has perfect information about the game and the second does not. The algorithm called PG-HSVI has been developed for these games, but it has a limited scalability. More scalable algorithms may be used, but they might not have convergence guarantees. To compare these computed strategies, a scalable best response algorithm may be used. The main goal of this work is to study One-Sided POSGs and their algorithms, formulate the best response of strategies as the problem of solving Partially Observable Markov Decision Process (POMDP), select appropriate solver to solve POMDP, experimentally evaluate the scalability, identify bottlenecks, and propose possible improvements to formulation/algorithm to improve scalability.

**Keywords:** Game, Game Theory, Value Iteration, HSVI, Monte Carlo Tree Search, POMCP, Markov Decision Process, Partially Observable MDP, PG-HSVI, Partially Observable Stochastic Games

**Supervisor:** Mgr. Branislav Bošanský, Ph.D.
Praha 2, Karlovo náměstí 13, E-407

## Abstrakt

Řešení částečně pozorovatelných, stochastických her (POSG) je výpočetně náročné, proto se výzkum soustředí na poddruhy POSG. Jednostranné POSG jsou hry pro dva hráče, jeden má perfektní informaci o hře, ale druhý ne. Algoritmus PG-HSVI byl vytvořen pro řešení těchto her, ale má pouze omezenou škálovatelnost. Je možné použít lépe škálovatelné algoritmy, ale ty negarantují konvergenci. Pro porovnávání vypočtených strategií může být použit škálovatelný algoritmus pro výpočet nejlepší odpovědi. Hlavní náplní této práce je nastudovat Jednostranné POSG a algoritmy pro jejich řešení, formulovat hledání nejlepší odpovědi na strategii jako Částečně pozorovatelný markovský rozhodovací proces (POMDP), vybrat vhodný algoritmus pro řešení POMDP, experimentálně ověřit škálovatelnost tohoto algoritmu, identifikovat překážky a navrhnout případné vylepšení formulace a algoritmy pro lepší škálovatelnost.

**Klíčová slova:** Hra, Teorie Her, Value Iteration, HSVI, Monte Carlo Tree Search, POMCP, Markovské rozhodovací procesy, Částečně pozorovatelné MRP, PG-HSVI, Částečně pozorovatelné stochastické hry

**Překlad názvu:** Algoritmy výpočtu nejlepší odpovědi ve stochastických hrách s neúplnou informací

# Contents

# Figures

# Tables

# Chapter 1

# Introduction

In most of the games, the main goal for each player is to win the game, therefore finding the best strategy to follow. To find and evaluate strategies based on given rules, games have to be mathematically defined. Dynamic Games can model many recreational games [Bar22] and also some real-world situations from fields of science like economics[Sam16], political science[Gut97], biology[KSJ15], security[LX13][BBB$^{+}$13] and more.

Markov Decision Processes are used to calculate a best response in these games, but this model requires for the game to be fully observable. However, Dynamic Games may often have only partial information. Markov Decision Processes may be extended to Partially Observable Markov Decision Processes (POMDPs), which corresponds to the type of Dynamic Games called Partially Observable Stochastic Games (POSGs). Most of the current techniques to solve these games focuses on games with only one player, which may, for example, simulate robot movement.

Solving Partially Observable Stochastic Games for more players in a general setting is intractable [HBP17], therefore the research focuses on subclasses like One-Sided POSGs, which assumes that one player has perfect information about the game, but the second one does not. These games may simulate hacker attack, where the hacker knows about the whole network and some defensive system periodically checks devices in this network. This subclass of POSGs may be solved to arbitrary precision using algorithm PG-HSVI [HBP17], which is a slightly modified version of the algorithm called Heuristic Search Value Iteration. This algorithm approximates the upper and lower bound, which may be systematically updated to reach the desired precision. However, this algorithm has a poor scalability. For example, in a situation where a hacker tries to invade network with 10 devices and some defensive system checks only one device at a time, there are millions of strategies and it may take a few days to find the optimal one. It is possible to modify the algorithm by changing the heuristics, but there may be no convergence guarantees. Without these guarantees, it is necessary to evaluate strategies independently on used heuristic.

The goal of this thesis is to extract strategies from the common interface used to solve POSGs. Formulate the best response evaluation of strategies as a problem of solving Partially Observable Markov Decision Process. Select appropriate solver for solving this POMDP. Evaluate scalability through experiments and also identify bottlenecks and propose possible improvements to formulation or algorithm in order to improve the scalability.

Chapter 2 defines Markov Decision Processes and Partially Observable Markov Decision Processes and shows ways to solve them. Chapter 3 describes Game Theory and games, which are used in this thesis. In Chapter 4 it is shown how to formulate POMDP, which would be used to compute a best response for perfect information player while having a fixed strategy for imperfect information player calculated by PG-HSVI algorithm for the same game. Chapter 5 contains implementation details of this approach and in the last chapter are results of this thesis and also few problems which were discovered during testing.

# Chapter 2

# Decision Making

Before defining games, which are usually played by more players, it is necessary to define the decision making of a single agent. Decision making for more players may be done if all players except one have fixed strategy, therefore the strategy is calculated for only 1 player. This chapter mainly focuses on Markov Decision Processes (MDPs), their extension Partially Observable MDPs (POMDPs), which deals with imperfect information about the game, and also the algorithms to solve them like Heuristic Search Value Iteration and Partially Observable Monte Carlo Planning. POMDPs are used to define a best response problem in Partially Observable Stochastic Games.

## 2.1 Markov Decision Processes

Markov decision process (MDP)[RN16] is a mathematical tool for stochastic decision making in a given situation. The basic understanding is that for a given situation, there are multiple actions to take, but each action may have different results with some probability. Mathematical representation of MDP is a tuple

$$(S, A, P(s'|a, s), R) \tag{2.1}$$

where $S$ are states, which defines the situation. $A$ are all possible actions actions, $P(s'|a, s)$ are probabilities, that taking action $a$ from $A$ in a state $s$ from $S$ will result in a state $s'$ from $S$. An important point of this is that probability only depends on state and action, which means that actions or states, which came beforehand do not affect it in any way. $R$ are all possible rewards, which evaluates profit.

Some states may have varying actions, therefore $A(s)$ is defined as a function which for given state $s$ returns possible actions. If the state does not have any actions, it is called terminal. Also, the reward differs based on state, action and state which is a result of this action. Function $R(s', a, s)$ returns reward from $R$, which is given after taking action $a$ from $A$ in state $s$ from $S$, while moving to state $s'$ from $S$.

A reward does not have to be dependent on all of these properties, sometimes reward depends only on $s'$. For this text, we will assume rewards are given after reaching a state. In this case, the average reward is

$$R(a, s) = R(s) + \sum_{s' \in S} P(s'|a, s) \cdot R(s') \quad s \in S, a \in A \tag{2.2}$$

where $R(s)$ is reward in current state, $a$ is action, $s$ is state.

Optimal action in state $s$ is then

$$\arg\max_{a \in A} R(a, s) \quad s \in S. \tag{2.3}$$

The average expected reward, which is not a guaranteed reward, but reward given on average after playing the same action in a given state $s$ may be written as

$$V(s) = R(s) + \max_{a \in A} \sum_{s' \in S} P(s'|a, s) \cdot R(s') \quad s \in S. \tag{2.4}$$

This updated value assumes, that player will take the best action. The notation is changed to $V(s)$ because this calculated value is not a guaranteed reward.

### 2.1.1 Rewards in larger MDPs

In the previous chapter, it was described how to calculate average rewards for a given state, action and rewards, but these calculations only have depth 1, this implies that only rewards which are "close" to the state are taken into account, this means it takes only 1 action to get this reward. In most of the games, it is necessary to perform more actions to get some reward.

One of the possible solutions to this problem is to instead of calculating with reward in a given state, to calculate with maximum average reward in the same state.

$$V(s) = R(s) + \max_{a \in A} \sum_{s' \in S} P(s'|a, s) \cdot V(s') \quad s \in S. \tag{2.5}$$

The difference between 2.5 and equation 2.4 is that reward $V(s)$ is a reward in adjacent states calculated through the same equation. This means that the calculation has to be done for all states. This method raises problem that neighbouring states affect each other, that suggests their value may have not been found exactly. It is proved that these values converge by simply repeating this calculation[RN16]. This method is called Value Iteration. Pseudo-code for possible implementation is in Algorithm 1

**Algorithm 1** Value Iteration

```
    Input
        S states in game
        A every possible action in every state
        P probability of resulting state based on initial state and taken action
        $R$ rewards in all States
        $\epsilon$ terminating condition, for change in 1 iteration
1: $V \leftarrow R$
2: repeat
3:     $\Delta \leftarrow 0$
4:     for $s \in S$ do
5:         $V_0(s) \leftarrow V(s)$
6:         $V(s) \leftarrow R(s) + \max_{a \in A} \sum_{s' \in S} P(s'|a,s) \cdot V(s')$
7:         $\Delta \leftarrow \max(\Delta, |V_0(s) - V(s)|)$
8: until $\Delta < \epsilon$
```

It is important to note that Value Iteration returns the optimal action for every given state. The reward value in Value Iteration is just expected average reward, not the guaranteed reward after every playthrough.

### 2.1.2 Discount Factor

While Value Iteration finds the average reward by playing the best strategy, it does not take into account that there may be one huge reward, which is far away and it is not feasible to get there in an acceptable time. So closer rewards should have more value than distant rewards. This may be done by adding multiplication constant $\lambda$, which is used to decrease the value of distant rewards. Value Iteration would then be

$$V(s) = R(s) + \lambda \max_{a \in A} \sum_{s' \in S} P(s'|a,s) \cdot V(s') \quad s \in S, \tag{2.6}$$

where $0 \leq \lambda \leq 1$. This constant is called the discount factor. If $\lambda = 1$, you get equation 2.5, if $\lambda = 0$, then

$$V(s) = R(s) \quad s \in S, \tag{2.7}$$

which says that only reward in a given state will be taken into account while calculating possible average reward.
MDP with discount factor becomes this tuple

$$(S, A, P(s'|a, s), R, \lambda) \tag{2.8}$$

## Grid world in 1D

Consider this one-player game where you are on a grid line with 3 states, left (L), middle (M) and right (R) as seen in Figure 2.1 and your only movement is to go left or right. You have 80% chance going the right way, 10% going the wrong way and 10 % not moving at all. The starting point is in the middle. Staying in middle will result in reward 0, going left has reward -10 a going right has reward 10. The game has only 1 turn.

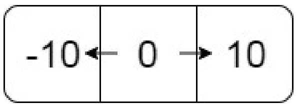

**Figure 2.1:** Graphical representation of 1D grid game

Probabilities are as given

$$P(\mathrm{L}|\leftarrow, \mathrm{M}) = 0,8 \tag{2.9}$$
$$P(\mathrm{M}|\leftarrow, \mathrm{M}) = 0,1 \tag{2.10}$$
$$P(\mathrm{R}|\leftarrow, \mathrm{M}) = 0,1 \tag{2.11}$$
$$P(\mathrm{L}|\rightarrow, \mathrm{M}) = 0,1 \tag{2.12}$$
$$P(\mathrm{M}|\rightarrow, \mathrm{M}) = 0,1 \tag{2.13}$$
$$P(\mathrm{R}|\rightarrow, \mathrm{M}) = 0,8 \tag{2.14}$$

Average rewards are

$$\begin{aligned} R(\leftarrow, \mathrm{M}) &= P(\mathrm{L}|\leftarrow, \mathrm{M}) \cdot R(\mathrm{L}) + P(\mathrm{M}|\leftarrow, \mathrm{M}) \cdot R(\mathrm{M}) + P(\mathrm{R}|\leftarrow, \mathrm{M}) \cdot R(\mathrm{R}) \\ &= -0,8 \cdot 10 + 0,1 \cdot 0 + 0,1 \cdot 10 = -7 \end{aligned} \tag{2.15}$$

$$\begin{aligned} R(\rightarrow, \mathrm{M}) &= P(\mathrm{L}|\rightarrow, \mathrm{M}) \cdot R(\mathrm{L}) + P(\mathrm{M}|\rightarrow, \mathrm{M}) \cdot R(\mathrm{M}) + P(\mathrm{R}|\rightarrow, \mathrm{M}) \cdot R(\mathrm{R}) \\ &= -0,1 \cdot 10 + 0,1 \cdot 0 + 0,8 \cdot 10 = 7 \end{aligned} \tag{2.16}$$

For this example it is obvious that $-7 < 7$, meaning that the best action is moving right.

### Larger grid world

If the game has larger world like Figure 2.2, then solving would be almost the same as 2.3.3. It is easy to see, that it in every state the best action is to move right.

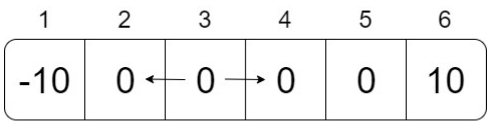

**Figure 2.2:** Graphical representation of larger 1D grid game

### 2.1.3 Monte Carlo Tree Search

MDPs may be solved even without the knowledge of the actual model[KS]. Solving may be done by simply taking actions from the initial state until the end, or to given finite horizon and keeping track on rewards. MDP is then transformed into a tree, which has the initial state as root, actions are edges leading to children nodes. For larger MDPs, the tree may be too large, so instead of generating the whole tree at the beginning, only the root is created. New nodes to the tree are added based on these steps.

1. Selection - Starting from root select childs according to heuristic, until leaf is reached. Leaf is not necessarily terminal state, but it is state from which the simulation was not yet started.
2. Expansion - If this leaf is not terminal, create child nodes for this state based on transition model.
3. Simulation - Play the game until terminal state is reached or depth of tree is reached. Simulation may be done according to some heuristic, or by just randomly taking actions.
4. Backpropagation - Propagates the reward recieved from newly expanded child up to the root with discount factor

Every node in the tree keeps the track on how often it was visited and the average reward from continuing strategy onward. To find an optimal strategy quickly, it is better to try actions which had previously the highest reward.

To avoid missing an optimal strategy, a heuristic may be applied to try some actions which were not tried for a long time.

**Algorithm 2** Monte Carlo Tree Search

```
    Input
        G game
        I amount of simulations
 1: $s_0 \leftarrow$ Initial State of Game
 2: $I_c \leftarrow 0$
 3: $S_V \leftarrow$ Empty List
 4: repeat
 5:     $I_c = I_c + 1$
 6:     $s_c \leftarrow s_0$
 7:     while $s_c \in S_V$ do
 8:         $s_c \leftarrow$ SelectChild($s_c$)
 9:     $r \leftarrow$ SimulateGame($s_c$)
10:     $s_f \leftarrow$ Expand($s_c$)
11:     $s_c \leftarrow s_f$
12:     BackpropagationToInitialState(($s_f, r$))
13: until $I_c < I$
```

## 2.2 Partially Observable Markov Decision Processes

MDPs mostly presents a way to find the optimal action and the average reward in perfect information games. MDP may be extended for imperfect information games. We will assume that this imperfect information is about the current state, meaning player will not know in which state he is, but he will have at least some information about it. This information is called observation. POMDP is then tuple

$$(S, A, P(s'|a, s), R, \lambda, \Omega, O(o|s, a, s')) \tag{2.17}$$

where $\Omega$ is set of all possible observations and $O(o|s, a, s')$ is probability of recieving an observation $o$ from $\Omega$ for given action $a$ from $A$ in state $s$ from $S$ with result state $s'$ also from $S$. Observations may represent almost anything, for example, observations in Example 2.3.4 might be, if a player is in an odd or even-numbered state, or how far is he from the closest terminal state. For this easy example, it is easy to see that if the probability of moving the right way is bigger than moving the wrong way, it is always better to go right, which is same as the strategy in perfect information case. Most of the time the strategy for perfect information game will be different then its imperfect information counterpart.

Every state has its set of observations and more states may have the same observations. Based on this, the player will rarely have information in which state he is, so he must assume all possible states based on observation. Therefore, the probability distribution over possible states is used, instead of states. This probability distribution is called belief and it is used instead of regular states to calculate optimal strategy. Unlike original states, belief depends on the history of previous actions and observations.

To solve this problem similarly to MDP, beliefs will be taken into account instead of states. Probability for initial belief is

$$b_0(s) = P(s) \quad s \in S \tag{2.18}$$

meaning that initial belief is just probability that the game is in a given state, without any additional information. The probability distribution over states for belief in after playing action $a$ and receiving observation $o$ is

$$b_t(s') = \frac{O(o|a,s') \sum_{s \in S} P(s'|a,s) b_{t-1}(s)}{\sum_{s'' \in S} (O(o|a,s'') \sum_{s \in S}^{n} P(s''|a,s) b_{t-1}(s))} \quad s' \in S. \tag{2.19}$$

This says that new belief probability that game is in different state $s'$ depends on taken action $a$ in belief state before action $b_{t-1}$ and received observation $o$. Numerator is basically probability of getting observation times probability of transition from all possible states, and their belief. Denominator is the same probability but for all possible states, which may have the same observation. This means that if game has only 1 state with given observation then

$$b_t(s') = 1 \tag{2.20}$$

Because beliefs may be seen as new states, transition model has to be changed accordingly

$$O(o|a,b) = \sum_{s' \in S} \left( O(o|a,s') \sum_{s \in S} P(s'|a,s) b_{t-1}(s) \right) \quad a \in A \tag{2.21}$$

$$P(b'|a,b) = \sum_{o \in \Omega} O(o|a,b) \quad a \in A \tag{2.22}$$

Observations $o$ are only observations which are plausible to occur while transitioning from belief state $b$ to $b'$ after taking action $a$.
Value iteration will then be

$$V(b) = \max_{a \in A} \left( \sum_{s \in S} R(a,s) b(s) + \gamma \sum_{o \in \Omega} O(o|a,b) \cdot V(\tau(b,a,o)) \right), \tag{2.23}$$

where $\tau(b,a,o)$ is transition, which returns new belief state after taking action $a$ in belief state $b$ and recieving observation $o$. The biggest problem with solving this transformed MDP is that there are infinitely many belief states.

### 2.2.1 Alpha Vectors

Expected rewards in POMDP are based on belief. This means that expected average reward differs continuously according to belief. Expected reward function $V(s)$ is then continuous too based on belief. This function is convex because it is made out of linear functions [SS73], these functions are called Alpha Vectors ($\alpha$-vectors). Single $\alpha$-vector represents a reward for fixed strategy in a state based on belief. Therefore the value function $V$ is

$$V = \{\alpha_0, \dots, \alpha_n\} \tag{2.24}$$

Expected highest reward for given action in some belief may then be written as:

$$V(b) = \max_{\alpha \in V} \sum_{s \in S} \alpha(s) b(s) \tag{2.25}$$

This equation may then be used to create value iteration for POMDP

$$V(b) = \max_{a \in A} \left( \sum_{s \in S} (R(s,a)b(s)) + \gamma \sum_{o \in \Omega} \max_{\alpha \in V} \sum_{s \in S} \sum_{s' \in S} P(s'|a,s) O(o|a,s') \alpha(s') b(s) \right) \tag{2.26}$$

Plots on Figure 2.3 and Figure 2.4 show how $\alpha$-vectors may look for some game. Also optimal reward is added as reference. Second plot has additional 1 $\alpha$-vector to show changes to expected reward.

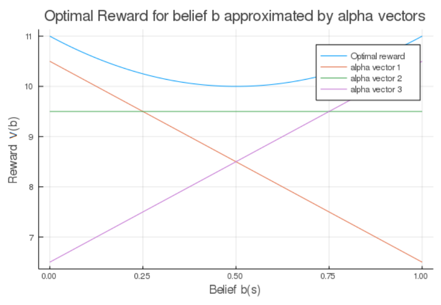

**Figure 2.3:** Expected reward using 3 alpha-vectors

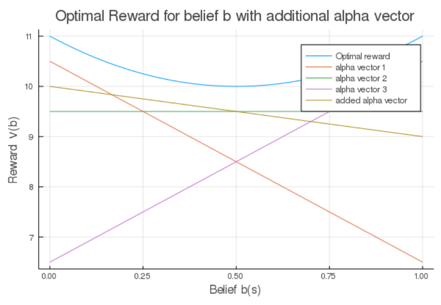

**Figure 2.4:** Expected reward after adding 1 alpha-vector

### 2.2.2 Heuristic Search Value Iteration

Solving POMDPs by using simple Value iteration is not scalable enough. More scalable algorithm is Heuristic Search Value Iteration (HSVI)[SS04] which approximates value as an upper and lower estimate. Main Upper estimate is made by using Value iteration from equation 2.23. The lower

estimate is then made by adding alpha-vectors and then using equation 2.26. The algorithm also implements heuristic, which finds the largest gap between lower and upper bound beliefs, and updates values there.
All alpha-vectors in lower bound are labelled as $V_L$

$$V_L = \{\alpha_0, \dots, \alpha_n\} \tag{2.27}$$

All points in upper bound are labelled as $V_U$.

$$V_U = \{V(b_1), \dots, V(b_m)\} \tag{2.28}$$

When started, HSVI initializes Upper and Lower bounds. Lower Bound is initialized by adding alpha-vectors which are just taking the same action over and over again. This means that the worst reward to possibly get is to take the same action every time. Upper bound is initialized by transforming POMDP to MDP, where every state corresponds to Action-observation-state tuple. After the initialization, the algorithm repeatedly calls a recursive method. In this method action and observation are selected. Action is greedily selected based on which action will have the highest expected reward. This may be written as

$$a_t = \arg\max_{a \in A} V_U(b, a) \tag{2.29}$$

Observation is then selected based of probability it will occur and gap between upper bound and lower bound for possible belief.

$$o_t = \arg\max_{o \in \Omega} \left(O(o|a_t, b)(V_U(\tau(b, a_t, o)) - V_L(\tau(b, a_t, o)) - \epsilon\gamma^{-(t+1)})\right) \tag{2.30}$$

Function $\tau(b, a, o)$ returns new belief based on current belief, taken action and recieved observation.

Update for lower bound is then adding new alpha-vector, which is computed for given belief $b$ using these equations

$$\alpha_{a,o} = \arg\max_{\alpha \in V_L}(\alpha\tau(b, a, o)) \tag{2.31}$$

$$\alpha_a(s) = R(s, a) + \gamma \sum_{o \in \Omega} \sum_{s' \in S} \alpha_{a,o}(s') O(o|a, s) P(s'|a, s) \tag{2.32}$$

$$V_L = V_L \cup \arg\max_{\alpha_a}(\alpha_a b) \tag{2.33}$$

$$V_U = V_U \cup \max_{a \in A} \left( \sum_{s \in S} R(a, s) b(s) + \gamma \sum_{o \in \Omega} O(o|a, b) \cdot V(\tau(b, a, o)) \right) \tag{2.34}$$

Keeping all points and alpha-vectors is redundant, because some are dominated by other ones, so these are continously removed. It is proved that HSVI converges to the optimal value.[SS04]

**Algorithm 3** Heuristic Search Value Iteration

```
    Input
        G game
        $\epsilon$ terminating condition,
 1: Initialize $V_L$, $V_U$
 2: repeat
 3:     EXPLORE($b, \epsilon, t$)
 4: until $V_U(b_0) - V_L(b_0) > \epsilon$
 5:
 6: function EXPLORE($b, \epsilon, t$)
 7:     if $V_U(b) - V_L(b) > \epsilon\gamma^{-t}$ then
 8:         $a, o \leftarrow$ select action and observation according to heuristic
 9:         EXPLORE($\tau(b, a_t, o_t), \epsilon, t + 1$)
10:         UPDATEBOUNDS($b$)
11:     return
```

### 2.2.3 Partially Observable Monte Carlo Planning (POMCP)

MCTS is used to solve games without the knowledge of the actual model, instead, it performs simulations of a given MDP. This approach may be used for partially observable MDPs[SV10]. Because the actual states are unknown, it is better to use histories as nodes of a tree. The root is just initial belief and edges in odd-numbered rows represents actions and in even-numbered rows represents observations. History of the child is then defined by the previous history, taken action and observation. Belief about the state depends on history. The algorithm then works as MCTS, where nodes use history instead of states.

On Figures 2.5, 2.6, 2.7, 2.8, 2.9 it is shown how POMCP searches through belief space. Every node in the tree contains value $N$, which is how many times this action-observation history occurred during simulations. Action nodes also contain average reward $R$ after taking this action at this point in the simulation. In the beginning, the node is selected according to the heuristic. POMCP uses equation

$$Q(h,a) = V(h,a) + c\sqrt{\frac{\ln N(h)}{N(h,a)}} \tag{2.35}$$

to select the next nodes until it gets to the leaf of the tree. This equation only takes actions into account, because POMCP cannot predict, which observation it will receive. Nodes are not representing states in the game, but the history of action and observation from the root. After POMCP gets to the leaf, it expands this node, by getting all possible actions. After that one action is taken and the game is simulated until the end. After reaching the terminal state, the reward is backpropagated from the new leaf up to the root. In this example there is no discount factor, otherwise, the reward is backpropagated with discount factor. One thing to note is that every action node has a higher $N$, then all sum of all its children $N$. When the new leaf is expanded, all action nodes are added, but there are no observations for them. From this new expansion, the game is finished, but only $N$ in action node is updated, and not in observation. This means every time new observation occurs its value $N$ is not updated.

POMCP usually needs a lot of iterations for the reward to converge. Advantage of this method is that it does not need to know how the POMDP is defined because it just simulates the actions. This means that size of state space does not affect much how long is one simulation. On the other hand, the longer the game lasts, the harder it is for the algorithm to find an optimal strategy. Also, POMCP does not guarantee to find the optimal strategy, but the found reward should at least be close to the optimal value. [SV10] (Theorem 1).

While using the strategy of imperfect information player, the state space gets exponentially larger. Compared to the HSVI, the POMCP is more scalable, because one iteration of the algorithm is almost not affected by state space, but only by the length of the game, this makes it for large state spaces faster than HSVI.

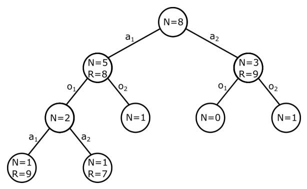

**Figure 2.5:** Tree before selection

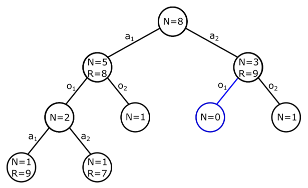

**Figure 2.6:** Select an action and then receive an observation for expanding

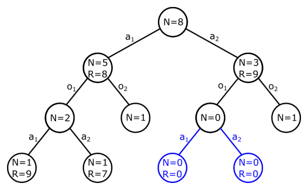

**Figure 2.7:** Expansion from current node

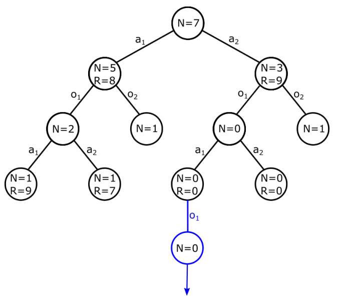

**Figure 2.8:** Select random action and simulate rest of the game

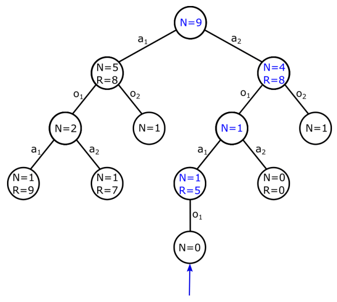

**Figure 2.9:** Backpropagate the reward for current simulation

# Chapter 3

# Solving One-Sided Partially observable stochastic games

The focus of this chapter is on the definition of a game in general and also a type of games used in this thesis which are Partially Observable Stochastic Games (POSGs) and its subtype One-Sided Partially Observable Stochastic Games (OS-POSGs), in which one player has perfect information and second has only partial information. These games may be solved for a player with imperfect information. This strategy accounts that the second player would play the best strategy, so it is the worst-case scenario for a player with imperfect information. Calculating this strategy is computationally difficult and algorithms with convergence guarantees are not scalable enough. To use more scalable algorithms, that do not have convergence guarantees, it is necessary to evaluate computed strategies. In this chapter, we define how to construct POMDP out of the original game and computed strategy for imperfect information player. This POMDP may then be solved to evaluate the computed strategy.

## 3.1 Game

Game is a mathematical description of strategic interactions, which are the rules defining the game. Every game has players, which participate in it. There may be one or more players in a single game, but usually games are for more players. Depending on the game, players take turns or they play simultaneously. The state is a description of a game in a given point, for example, in chess, it is a position of all figures and which player plays next. Possible states may vary for different players. Action is a transition between 2 states (both states may be the same) which may result in some reward. The goal in every game is to collect the highest reward. Stochastic Game is defined as a tuple

$$(P, S, A, R, T(a, s), S_s, S_t) \tag{3.1}$$

where $P$ are players in the game, $S$ states, $A$ actions, $R$ are rewards, $T(a, s)$ is a function which returns new state with some probability after playing action $a$ from $A$ in a state $s$ from $S$, $S_s$ is starting state, $S_t$ are all terminal states, where the game ends. Some games may require more parameters.

## 3.2 One-Sided Partially Observable Stochastic Games

In Partially Observable Stochastic Games players do not have the full information about the game, so every player has to make a decision based on some partial information, which he receives. POSGs are stochastic, so taking one action in the same state, may have different results, this is similar to MDPs. Because solving POSGs is generally intractable, the research focuses on subclasses, which are not as general. In this thesis One-Sided POSGs are used, these games are two-player and zero-sum, that means both players play against each other and when one player receives a reward, second will receive the same amount as a negative reward. Solving the game, while both players would have only partial information would be difficult and maybe even impossible, so another important property of these games is that one player has perfect information about the game. These changes make solving these POSGs much easier. In these games, both players choose actions to play at once.

## 3.3 Extracting strategy

The game is defined as tuple

$$(S_G, A_1, A_2, \Omega, P(s'_G, o|s_G, a_1, a_2), R(s_G, a_1, a_2), \gamma) \tag{3.2}$$

where $S_G$ are all possible states, $A_1$ are actions for imperfect information player, $A_2$ are action for perfect information player, $\Omega$ are all possible observations for imperfect information player, $P(s'_G, o|s_G, a_1, a_2)$ is probability that after playing actions $a_1$, $a_2$ from $A_1$, $A_2$ respectively in a state $s_G \in S_G$, the resulting state would be $s'_G \in S_G$ and imperfect information player will receive observation $o$ from $\Omega$ and $R(s_G, a_1, a_2)$ is reward given to imperfect information player after playing actions $a_1$ and $a_2$ from $A_1$ and $A_2$ in a state $s_G$ from $S_G$ and $\gamma$ is discount factor. When the game is solved, the strategy for imperfect information player has to be extracted from the data structures used by the PG-HSVI algorithm. HSVI computes upper and lower bounds separately. Only lower bound is extracted and its calculated strategy is made out of alpha-vectors. Every alpha-vector contains probability distribution over all possible actions in calculated strategy. Even though alpha-vectors are linear functions, the probability distribution over actions in alpha-vector does not change based on belief, this means that alpha-vectors may be viewed as a single strategic point in the game and not as a function. These points are called Strategy Nodes. Strategy node corresponds to the part of belief space where single alpha-vector is selected to be played. Playing an action changes belief, therefore it changes the Strategy Node. All Strategy Nodes with an included probability distribution over actions and possible Strategy node transitions have to be exported. Also, the initial belief, which corresponds to exactly one Strategy Node, has to be exported.

## 3.4 Defining problem as POMDP

To find strategy for perfect information player, by using strategy for imperfect information POMDP is created out of game definition and extracted Strategy nodes from the calculated strategy for the same game. Extracted Strategy is defined as a tuple

$$(SN, A_1(sn), \rho(a_1|sn), \sigma(sn'|o, a_1, sn)) \tag{3.3}$$

where $SN$ are all possible Strategy Nodes, $A_1(sn)$ are all possible actions in given Strategy Node $sn$ from $SN$, $\rho(a_1|sn)$ are probabilities for taking action $a_1$ from $A_1$ in Strategy Node $sn$ from $SN$ taken from probability distribution over actions in alpha-vector. $\sigma(sn'|o, a_1, sn)$ is the transition model between strategy nodes, which is probability that after taking action $a_1$ from $A_1$ in Strategy Node $sn$ from $SN$ and receiving observation $o$ from $\Omega$, the resulting Strategy Node would be $sn'$ from $SN$.

POMDP is then defined as

$$S = \{(s_1, s_2); s_1 \in S_G, s_2 \in SN\} \tag{3.4}$$

$$A = A_2 \tag{3.5}$$

$$\Omega = \{s_1; s_1 \in S_G\} \tag{3.6}$$

$$P((s_1', s_2')|a, (s_1, s_2)) = \sum_{a_1 \in A_1} \sum_{o \in \Omega} \sigma(s_2'|s_2, a_1, o) \cdot P(s_1', o|s_1, a, a_1) \cdot \rho(a_1|s_2)$$
$$s_1, s_1' \in S_G, s_2, s_2' \in SN, a \in A \tag{3.7}$$

$$R((s_1, s_2), a) = R(s_1, a_1, a_2) \quad a, a_2 \in A, a_1 \in A_1 \tag{3.8}$$

States of this POMDP are all possible combinations between original states and strategy nodes from HSVI result. These new states are called the Joint States and every Joint State is uniquely defined by the original state and strategy node. Actions are just original actions for perfect information player because for this player we try to find the optimal strategy. Strategy for imperfect information player is already calculated. Observations in this POMDP are states from the initial game. Probability of transitioning from state $S$ to state $S'$ after playing action $A$ depends on the probability that imperfect information player will take action $a_1$ in calculated strategy in given strategy node and probability of this transition taken from original game. Rewards are taken from original definition, but it requires to know action taken by imperfect information player.

## Example

Assume a game for 2 players, in which both players have only two actions. Player one receives a reward if both player play the same action, otherwise, the player two receives a reward. Both players take their turns simultaneously. The game lasts only one round. This game has only 2 states $s_1$, $s_t$, which are the start of the game and the terminal state respectively. The game has only 1 observation $o$, which is given regardless of action. Strategy for the second player may look like Figure 3.1

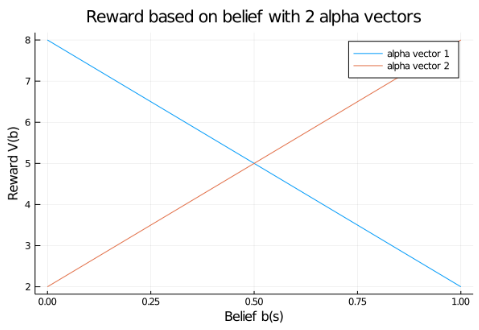

**Figure 3.1:** Example strategy with 2 alpha-vectors

This strategy consists of 2 alpha-vectors. Every strategy node corresponds to one alpha-vector, therefore the strategy contains 2 strategy nodes $sn_1, sn_2$.

Probability distributions over actions in these nodes are

$$\rho(a_0|sn_1) = \rho(a_1|sn_2) = 0.8 \tag{3.9}$$
$$\rho(a_1|sn_1) = \rho(a_0|sn_2) = 0.2 \tag{3.10}$$

This game consists of only one round, therefore every action results in a terminal state, which is also a strategy node $sn_t$, that does not have any actions available

$$\sigma(sn_t|sn_1, a_1, o) = \sigma(sn_t|sn_2, a_1, o) = \sigma(sn_t|sn_1, a_2, o) = \sigma(sn_t|sn_2, a_2, o) = 1 \tag{3.11}$$

POMDP which will be used to find a best response for player 1 is then defined like

$$S = \{(s_1, sn_1), (s_1, sn_2), (s_t, sn_t)\} \tag{3.12}$$

$$A = \{a_1, a_2\} \tag{3.13}$$

$$\Omega = \{s_1, s_t\} \tag{3.14}$$

$$\begin{aligned} P((s_t, sn_t)|a_1, (s_1, sn_1)) = P((s_t, sn_t)|a_1, (s_1, sn_2)) = \\ P((s_t, sn_t)|a_2, (s_1, sn_1)) = P((s_t, sn_t)|a_2, (s_1, sn_2)) = 1 \end{aligned} \tag{3.15}$$

$$R(s_1, a_1, a_1) = R(s_1, a_2, a_2) = 10 \tag{3.16}$$

$$R(s_1, a_1, a_2) = R(s_1, a_2, a_1) = -10 \tag{3.17}$$

In this game, it does not matter if the player has perfect information because it lasts only one round.

# Chapter 4

# Implementation

In the previous chapter, it was proposed how to construct POMDP out of generally defined OS-POSG and calculated strategy for imperfect information player. Solving this POMDP should account for the optimal strategy for perfect information player. However, the program for solving OS-POSGs, which uses HSVI algorithm is already developed and the proposed solution has to be implemented to work with this already developed interface. This newly created POMDP was solved, by using an already implemented algorithm Partially Observable Monte Carlo Planning from Stanford University [ESB+17]. This solution is in programming language Julia and is part of official Julia modules. This chapter contains specific details about the implementation of the proposed solution and examples of OS-POSGs, which were then used for testing. To reduce memory requirements in HSVI algorithm, sets of states called partitions are used. Partitions reduce the number of transitions in the game. Time requirements in POMCP were reduced by using model, which for given state and action returned new state, observation and reward.

## 4.1 Partitions

Partition is a collection of states, which meets the requirement, that every partition has at least one state and that every state must be in exactly one partition. Both players know all the time in which partition they are. Playable actions for Imperfect information player depend only on the current partition. Lower and upper bounds solved by HSVI are then solved per partition. Every State and Alpha-Vector contains information about in which partition they are, this reduces the number of possible states in POMDP, because the additional requirement is

$$S = \{(s_1, s_2); s_1 \in S_G, s_2 \in SN, Pa(s_1) = Pa(s_2)\} \tag{4.1}$$

where $Pa(s_1)$ is partition for state $s_1$ and $Pa(s_2)$ is partition for strategy node $s_2$.

## 4.2 Transition model

Transition, observation and reward model were all merged into one in method, which for given Joint State and action return resulting Joint State, observation and reward. This function is defined as

$$(s', o, R) = Gen(s, a). \tag{4.2}$$

For a given state $s$ from $S$ and action $a$ from $A$ a result state $s'$ from $S$, Observation $o$ from $\Omega$ and Reward $R$ are returned based on transitions in POMDP. Random Number Generator is used to choose which tuple to return based on probabilities. Mersenne Twister is used for this random generation.

This model firstly iterates through all possible transitions for the state from the original game, which are held in Joint State. It finds all transitions for given action $a$. Then the algorithm iterates through all transitions in a given strategy node. For all transitions, which have matching imperfect information actions $a_1$ from $A_1$, observation $o$ from $\Omega$ the new Joint States are created, which holds information about original state and strategy node after possible transitions with probability, which is calculated as follows

$$P(s'|s,a) = \sum_{a_1 \in A_1} \sum_{o \in \Omega} \sigma(s'_2|s_2, a_1, o) \cdot P(s'_1, o|s_1, a, a_1) \cdot \rho(a_1|s_2) \quad s, s' \in S, a \in A \tag{4.3}$$

where $s_1$ and $s_2$ are game state and strategy node respectively taken from joint state $s$. Sometimes these probabilities do not sum up to 1, so it is necessary to normalize them

$$P_n(s'|s,a) = \frac{P(s'|s,a)}{\sum_{s'' \in S} P(s''|s,a)} \quad s, s' \in S, a \in A \tag{4.4}$$

## 4.3 Output

The result from Julia is a whole Monte Carlo tree from POMCP, where every even row represents taken action, which also contains average reward for a given action and every odd row contains observations after the previous action. This tree may look like Figure 4.1

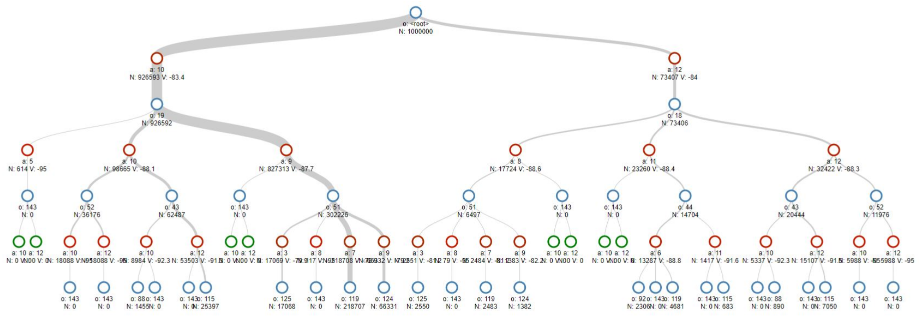

**Figure 4.1:** Tree result from POMCP for smaller game

This tree has only 6 rows shown, just to make the image more clear, but the tree is larger. Every action node has information about how many times the action was taken, the index of this action and the average reward. On the other hand, observation contains information about how many times it occurred and observation index. The whole tree is transformed into an HTML file and may be opened by any internet browser.

## 4.4 Experiment Domains

The used algorithm requires specifically designed games to be transformed from a two-player game into POMDP. Every game has to be defined as a tuple

$$(S, Pa, A_1, A_2, \Omega, P(s', o|s, a_1, a_2), R(s, a_1, a_2), \gamma) \tag{4.5}$$

where $S$ are all possible states, $Pa$ are partitions in the game. $A_1$ are actions for player one, which has only imperfect information. Therefore his actions are only dependent on a partition. $A_2$ are actions for the player with perfect information and differs according to state. $\Omega$ is set of all possible observations $O$ in the game. $P(s', o|s, a_1, a_2)$ is transition and observation model merged into one. It is the probability that for a given state $s$ from $S$ and actions $a_1$, $a_2$ from $A_1, A_2$ respectively the game will result in state $s'$ from $S$ and imperfect information player would get observation $o$ from $\Omega$. In most used games for testing, this probability was equal to 1. $R(s, a_1, a_2)$ is reward model, that returns reward for the given state $s$ from $S$ and actions $a_1, a_2$ from $A_1$, $A_2$. $\gamma$ is a discount factor of a game.

All used games are defined on the graph. It is easy to scale the game, by just changing the graph on which the game is played.

There are no stochastic events in any of used domains, meaning

$$P(s', o|s, a_1, a_2) = 1 \tag{4.6}$$

Because both players know in which partition they are, the initial belief is defined as the initial partition and probability over states in this partition.

Games are created and saved using the Scala program, which also checks if the game is valid. For example, if every possible transition has defined reward, or if action-observation history results always in the same partition.

### 4.4.1 Pursuit Evasion Games

The first type of games used to test if algorithms worked properly was Pursuit Evasion Games.[ARS+03] Games of this type are for multiple players. Players are distinguished as pursuers and evaders. Pursuer tries to catch evader in a game, while evader tries to avoid being caught. All players take actions simultaneously, and their possible actions may not be the same. Games are played on graph and state depend on the position of both players. Reward when the pursuer catches the evader is $R = 95$ otherwise, it is $R = 0$.

### 4.4.2 Patrolling Games

These games [RMY+05] are played by two players, defender and attacker. While the defender moves through the graph, the attacker may invade any vertex and attack it. When the attacker commits to attacking some vertex, he cannot interrupt this attack and it lasts more than one round. If the defender finds which vertex was attacked, he wins, otherwise the attacker wins. This means the goal of the defender defender is to visit every vertex as often as possible, but without any predictable pattern.
When the attack is successful, the defender receives a negative reward, otherwise, the reward is

$$R = 0 \tag{4.7}$$

Every vertex may have a different reward if the attack is successful.

The defender does not know when and where the attacker starts the attack. On the other hand, the attacker has perfect information about the game.

### 4.4.3 Blocking Games

Blocking games were specifically created for this thesis. Two players play this game, where one moves through the graph and the second player is outside the graph and tries to predict his movement and block it. To make these games easy to generate and make general actions, game graphs follow a specific structure. There is always a fixed starting point in the middle of the graph. The graph is divided into rows, which are partitions in the game. Every vertex in a row is connected to $n$ vertices from the next row, which are called children and exactly one vertex from the previous row, which is called the parent. The graph has only $m$ rows and vertices in the last row do not have any children and vertex in the first row vertex does not have a parent. To make more possible routes to the end, vertex in a row is connected with two other neighbours in the same row.

If all vertices in the last row were terminal, the player would just take random actions to move forward until he gets to the end, so only $k$ vertices in the last row are terminal and give the player some reward. There must be more than one terminal state, otherwise, the game would be impossible to complete.

Because every vertex, which is not in the last row, has $n$ children, the player who moves through graph has $n$ possible actions to move forward, one action to move backwards and two actions to move sideways in the same row. The player who blocks movement, have $n$ actions to block any possible move forward, one action to block moving backwards to parent and one action to block any movement in the same row.

The player with imperfect information is the one, who blocks movement and his possible observations are: player moved to next row, the player moved to the previous row, the player moved in the same row, the player was blocked, the player moved to the terminal state.

It is easy to generate such games, but they scale fast by adding more rows and children. The generator was created, which takes as input amount of children $n$, the amount of row $m$ and the number of terminal states $k$. Game is generated with randomly spread terminal states in the last row.

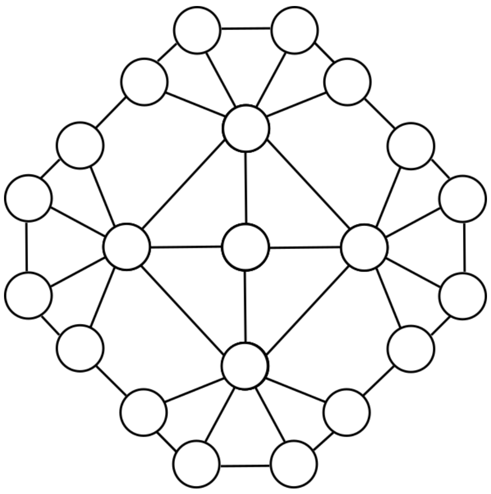

**Figure 4.2:** Graph for blocking game with 3 rows and 4 children

## 4.5 Example

Pursuit Evasion Game on a fully connected graph with three vertices is defined by seven distinct states. Six of those states are for possible positions for both players on the graph. The last state is when both players are in the same vertex, which is the terminal state of the game. Both players have the same actions, which are moves to any vertex. Partitions are defined as the position of imperfect information player. Only two observations are in the game, which only symbolizes if the game reached terminal state or not.

File with the definition of this game is shown in Appendix A. The game was solved using HSVI algorithm and rewards in lower and upper bounds were approximated as:

$$V_L = 86.36364 \tag{4.8}$$

$$V_U = 86.46228 \tag{4.9}$$

The output text file with description is shown in Appendix B. Only four alpha-vectors were used to approximate strategy, and all of them were taken from HSVI. Every partition has only one strategy node. The first three strategy nodes have the same probability for every action. This is an expected result because there is no strategy, which is superior, so moving randomly yields in highest average reward. Fourth strategy node is in a terminal state.

To use the POMCP, the original game and results have to be converted to POMDP. This POMDP has state space is defined as the Cartesian product of game states and strategy nodes. This number is reduced because states and strategy nodes have to have matching partitions. For this particular game, the state space is $7 \cdot 4 = 28$ without taking partitions into account or 7 with partitions. Actions for this POMDP are actions of perfect information player because its strategy we try to find. Observations are defined as game states, which does not contain strategy nodes. Transitions are separately calculated for game state and strategy node. Game states have transitions defined in original definition and strategy nodes have transitions defined as a probability distribution over possible actions and a probability distribution over different strategy nodes based on taken action. Rewards are defined in the game definition, so they only take game states without strategy nodes and actions into account. The initial state of this POMDP is one strategy node taken from HSVI result and a probability distribution over states in one partition used from the game definition. Discount factor remains the same as for the original game. One million tree iterations are used to calculate strategy. The average calculated reward is

$$V = 86.37188 \tag{4.10}$$

Reward from POMCP has an opposite sign as a reward from HSVI because it is calculated for the other player. Output tree from POMCP is shown at Figure 4.3.

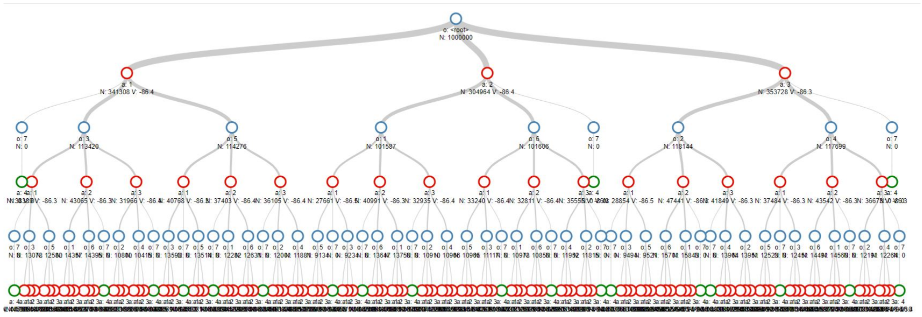

**Figure 4.3:** Tree result from POMCP for example

# Chapter 5

# Results

This chapter contains results of these tests and also states several problems encountered during these tests. Because POMCP is stochastic solver, results may vary everytime simulation is started, but after many iterations correct strategy POMCP converges to optimal value. Every game was solved in POMCP multiple times, but only values from the last tests are shown in this work however, the values were less than $10^0$ apart.

## 5.1 Pursuit Evasion Games

First games to be tested were Pursuit Evasion Games. Details of four used games, ordered by size are shown in Table 5.1.

| Game | States | Partitions | Actions of imperfect information player | Actions of perfect information player | Transitions |
|---|---|---|---|---|---|
| PEG1 | 143 | 21 | 145 | 13 | 2671 |
| PEG2 | 363 | 37 | 290 | 18 | 8123 |
| PEG3 | 731 | 57 | 485 | 23 | 18335 |
| PEG4 | 1299 | 82 | 730 | 28 | 34807 |

**Table 5.1:** Parameters for different Pursuit Evasion Games, which were tested

In Table 5.2 are shown results from HSVI and in Table 5.3 are results from POMCP

| Game | Strategy nodes | Lower HSVI Bound | Upper HSVI Bound | HSVI Time |
|---|---|---|---|---|
| PEG1 | 3757 | 83,48894 | 83,5877 | 4 minutes |
| PEG2 | 16734 | 77,71581 | 77,81509 | 1 hours |
| PEG3 | 69087 | 71,90179 | 72,90118 | 7 hours |
| PEG4 | 199453 | 65,74002 | 70,73929 | 68 hours |

**Table 5.2:** Results from HSVI for Pursuit Evasion Games

| Game | Tree Iterations | POMCP Reward | POMCP Time |
|---|---|---|---|
| PEG1 | 100 million | 83,19077 | 60 minutes |
| PEG2 | 100 million | 77,36696 | 95 minutes |
| PEG3 | 100 million | 70,95893 | 130 minutes |
| PEG4 | 100 million | 66,85947 | 165 minutes |

**Table 5.3:** Results from POMCP for Pursuit Evasion Games

For all tests 100 million POMCP simulations were used, to ensure that reward converges to the correct value, but 10 million might be sufficient enough. POMCP rewards were negative, because HSVI calculates reward for imperfect information player and POMCP to perfect information player. Expected results were that POMCP will converge to Lower bound from HSVI, but they seem to differ a bit.

In Figure 5.1 are shown changes to expected reward based on number of iterations.

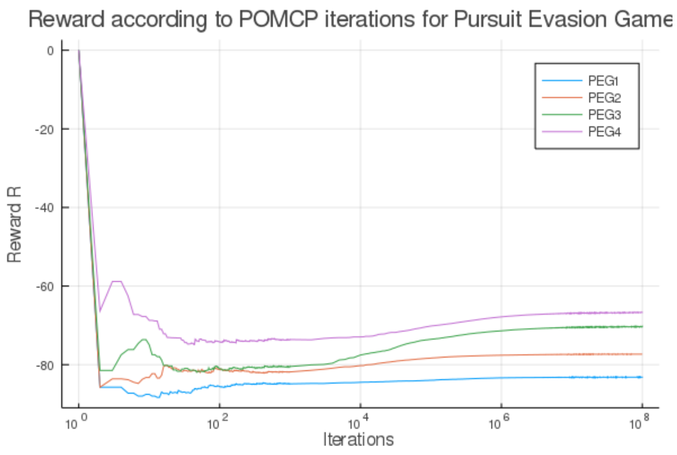

**Figure 5.1:** Rewards for Pursuit Evasion Games

## 5.2 Patrolling Games

Patrolling games are usually shorter then Pursuit Evasion Games. Five different Patrolling Games were used for tests and they are defined in Table 5.4

| Game | States | Partitions | Actions of imperfect information player | Actions of perfect information player | Transitions |
|---|---|---|---|---|---|
| Patroll 1 | 134 | 8 | 16 | 8 | 599 |
| Patroll 2 | 342 | 12 | 30 | 12 | 1975 |
| Patroll 3 | 482 | 14 | 37 | 14 | 2951 |
| Patroll 4 | 646 | 16 | 49 | 16 | 4699 |
| Patroll 5 | 834 | 18 | 56 | 18 | 6139 |

**Table 5.4:** Parameters for different Patrolling games, which were tested

Tables 5.5 and 5.6 contains information about results from HSVI for imperfect information player and POMCP for perfect information player respectively

| Game | Strategy nodes | Lower HSVI Bound | Upper HSVI Bound | HSVI Time |
|---|---|---|---|---|
| Patroll 1 | 942 | -51,79765 | -51,69773 | 5 minutes |
| Patroll 2 | 1722 | -49,30796 | -48,31187 | 20 minutes |
| Patroll 3 | 2401 | -57,57143 | -56,57440 | 30 minutes |
| Patroll 4 | 3765 | -60,9512 | -59,90201 | 1 hour |
| Patroll 5 | 4539 | -58,99490 | -57,99520 | 3 hours |

**Table 5.5:** Results from HSVI for Patrolling Games

| Game | Tree Iterations | POMCP Reward | POMCP Time |
|---|---|---|---|
| Patroll 1 | 10 million | -51,74353 | 4 minutes |
| Patroll 2 | 10 million | -48,39643 | 4 minutes |
| Patroll 3 | 10 million | -56,63545 | 5 minutes |
| Patroll 4 | 10 million | -59,97149 | 5 minutes |
| Patroll 5 | 10 million | -58.09309 | 5 minutes |

**Table 5.6:** Results from POMCP for Patrolling Games

Figure 5.2 shows changes to expected reward based on number of iterations.

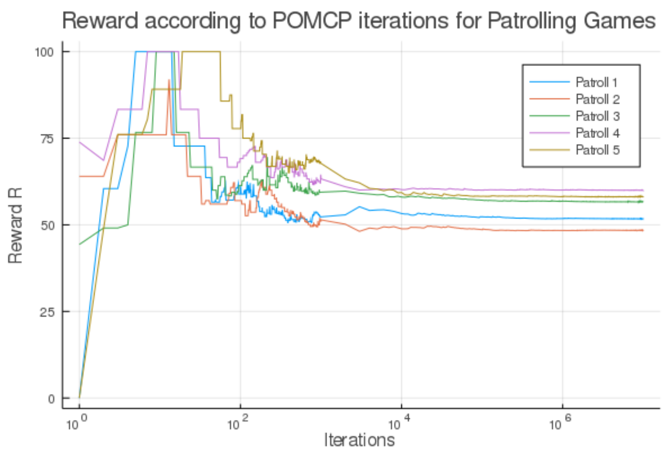

**Figure 5.2:** Rewards for Patrolling Games

## 5.3 Blocking Games

Three different blocking games were used for testing. Parameters of these games are shown in 5.7 and 5.8

| Game | States | Partitions | Actions of imperfect information player | Actions of perfect information player | Transitions |
|---|---|---|---|---|---|
| Block 1 | 16 | 4 | 8 | 8 | 736 |
| Block 2 | 35 | 5 | 7 | 7 | 1225 |
| Block 3 | 26 | 6 | 6 | 6 | 626 |

**Table 5.7:** Parameters for different Blocking games, which were tested

| Game | Rows | Children | Terminal States |
|---|---|---|---|
| Block 1 | 3 | 4 | 6 |
| Block 2 | 4 | 3 | 6 |
| Block 3 | 5 | 2 | 6 |

**Table 5.8:** Definition of used Blocking Games

In tables 5.9 and 5.10 are details from both HSVI and POMCP.
Figure 5.3 displays how average reward changes based on POMCP iterations.

| Game | Strategy nodes | Lower HSVI Bound | Upper HSVI Bound | HSVI Time |
|---|---|---|---|---|
| Block 1 | 942 | -58,39558 | -57,39725 | 4 minutes |
| Block 2 | 1722 | -34,85746 | -32,85776 | 3 days |
| Block 3 | 2401 | 2,80486 | 3,80027 | 6 hours |

**Table 5.9:** Results from HSVI for Blocking Games

| Game | Tree Iterations | POMCP Reward | POMCP Time |
|---|---|---|---|
| Block 1 | 10 million | -59,48058 | 9 minutes |
| Block 2 | 10 million | -36,56410 | 11 minutes |
| Block 3 | 10 million | 2,27276 | 40 minutes |

**Table 5.10:** Results from POMCP for Blocking Games

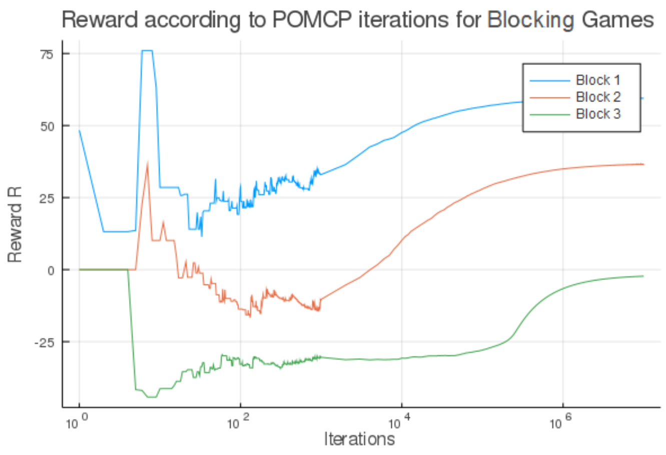

**Figure 5.3:** Rewards for Blocking Games

These games took longer time to finish the same amount of iterations as Patrolling and Pursuit Evasion Games. This happens due to the nature of these games, it usually takes more actions by perfect information player to get to the terminal state, even after playing the perfect strategy.

## 5.4 Issues with this approach

Even though most of the results match expectations, there were few problems, which had to be solved out, to ensure, that program is working correctly and results match the expectations.

### 5.4.1 Actions without transition

POMCP chooses, which action to take from a sample of all possible actions. This caused issues because possible actions depend on the state of the game. The algorithm also has a method, which returns actions based on history, but it has a bug, which causes it to call this method with only previous action and observation. From this information, it is not possible to find new possible actions. Because solver is implemented as an official Julia module, the fix should not affect the original code. So the first possible solution was to return terminal state with negative reward, which was smaller then the lowest possible reward in the game if the transition model was called with action, which was not possible for a given state. Other solution was to return the same state, but with some negative reward. Both of these solutions did not solve the problem, but just bypassed it. They also affected the final reward. In every state, there are more impossible than possible actions, so in simulations, the impossible action was chosen in more than 50% simulations. This made the algorithm to converge slowly. So another solution has to be found. Because both players know in which partition they are in any given point, the transition model was changed to return partition as observation, but perfect information player may have different actions in the same partition because its actions depend on the state. This caused that the problem still occurred, but it was reduced to happen in less than 20% simulations in large games. In smaller games, it occurred in less than 5% simulations. So observations were changed from partitions to states from the original game. The only state was returned and not a joint state, so perfect information player still did not have information about strategy node. This did not change the definition of game, because perfect information player knows in which state the game is before every action. Based on last action-observation pair, it was now possible to return possible actions.

### 5.4.2 Bug in POMCP with getting possible actions

After changing the method to return possible actions from received action-observation, another implementation bug was found, which caused that almost every other method call was without action-observation pair. This caused the program to crash. Problem was that the POMCP solver is part of a bigger module in Julia, which is implemented for multiple solvers. Random policy, which is used to explore new nodes, calls method without history, because some solvers, which are used for MDPs, do not require it.

This was solved by saving the current state in the game after every transition. This way even if the method was called without history, it was possible to return possible actions. It would seem that returning current state as observation is obsolete, but POMCP asks for possible actions only when expanding node and then saves them into memory for time optimization. This means that for the same history it does not find actions more than once. For perfect information player action-observation history is not sufficient to find possible actions, because they may differ for every state.

### 5.4.3 Missing transition for strategy nodes

Even after resolving two previous issues, sometimes transition function was called with a joint state-action combination which had only one strategy node transition. This issue occurred only one action before the end of the game. This means that strategy for imperfect information player expected that the game would end in next turn and therefore forced which action should perfect information player play in POMCP. In Patrolling games this behaviour is expected but in Pursuit Evasion and Blocking games not. This issue was not resolved, but it was bypassed by moving player terminal state with a lower reward than the lowest in the game. This ended the game even with different action by perfect information player. This is not a correct solution, but since this issue occurred on average on 1 in 100000 iterations, it did not affect the final results. A possible solution which would be more optimal is to have some default strategy for imperfect information player, which would be followed in this case.

# Chapter 6

# Conclusions

In this work, we defined how to solve game for perfect information player, while using compute strategy for imperfect information player in OS-POSGs. We formulated this problem as POMDP. we used POMCP algorithm because it is more scalable than HSVI, which may be seen in all three domains used in testing, even though in POMCP is exponentially bigger state space. This approach for solving the game for perfect information player was tested on results from PG-HSVI, which has convergence guarantees. From testing both Pursuit Evasion Games and Patrolling games, it is shown that the size of these games almost does not affect the time required for POMCP to converge. The largest PEG game had iterations almost 3 times longer the smallest, but the state space was 10 times larger in the original game and 500 times larger in POMDP with imperfect information player strategy. On the other hand size of patrolling games did not affect the length of iteration, mostly because all tested games had the same length of the attack. Blocking games which are usually longer and may take more than 15 actions to reach terminal state converged slower and iterations took longer. On average value converged after $10^7$ iterations, whereas PEG and Patrolling games converged between $10^5$ and $10^6$ POMCP iterations. Overall the results from POMCP corresponds to results from HSVI, therefore the found strategy for imperfect information player truly guarantees rewards between lower and upper bound.

In presented solutions, the most substantial bottleneck is in implementation of POMCP itself, but for this work the speed of this implementation is sufficient. To further increase the speed of POMCP it is possible to use multi-threading or better POMCP constant tuning.

## 6.1 Further work

Using POMCP algorithm to evaluate strategies for imperfect information player was successful however, only one version of HSVI, which guarantees to find the optimal strategy, was used for testing. The main goal of this approach was to evaluate strategies from different versions of HSVI while using different heuristics without any convergence guarantees. These different versions are yet to be tested in future.

This work may be extended by swapping which player has the perfect information and calculate strategy for the second player. This results in two different strategies, where either player 1 or 2 has the perfect information. These strategies may then be compared to see how much average reward changes based on which player has the perfect information.

# Appendix A

# Game Definition

```
7 4 4 4 2 55 55 0.9500  //Game definition
Position_1_2 0          //State name - Partition index
Position_1_3 0
Position_2_1 1
Position_2_3 1
Position_3_1 2
Position_3_2 2
endState 3
moveTo1_P1              //Actions for Imperfect
moveTo2_P1              //information player
moveTo3_P1
endAction_P1
moveTo1_P2              //Actions for Perfect
moveTo2_P2              //information player
moveTo3_P2
endAction_P2
game_continues          //Observations
endObs
0 1 2                   //Possible Perfect Information
0 1 2                   //player Actions for state
0 1 2
0 1 2
0 1 2
0 1 2
3
0 1 2                   //Possible Imperfect Information
0 1 2                   //player Actions for partition
0 1 2
```

```
3
3 2 0 0 4 1.000000  //Transitions defined as:
4 0 2 0 1 1.000000  // Initial state
2 2 0 0 4 1.000000  // Imperfect Information Player action
1 0 1 0 0 1.000000  // Perfect Information Player action
0 1 0 0 2 1.000000  // Recieved Observation
0 0 2 0 1 1.000000  // Resulting State
3 1 0 0 2 1.000000  // Probability
1 2 2 1 6 1.000000
5 2 1 0 5 1.000000
4 1 2 0 3 1.000000
5 2 2 1 6 1.000000
2 2 2 1 6 1.000000
2 1 0 0 2 1.000000
3 0 1 0 0 1.000000
0 1 2 0 3 1.000000
2 2 1 0 5 1.000000
5 1 1 1 6 1.000000
4 0 1 0 0 1.000000
1 1 1 1 6 1.000000
0 2 2 1 6 1.000000
2 1 1 1 6 1.000000
5 2 0 0 4 1.000000
1 0 2 0 1 1.000000
3 0 2 0 1 1.000000
3 2 2 1 6 1.000000
1 2 0 0 4 1.000000
0 2 0 0 4 1.000000
6 3 3 1 6 1.000000
4 2 0 0 4 1.000000
2 0 1 0 0 1.000000
4 2 2 1 6 1.000000
5 0 2 0 1 1.000000
0 1 1 1 6 1.000000
3 0 0 1 6 1.000000
4 1 1 1 6 1.000000
0 0 0 1 6 1.000000
5 1 2 0 3 1.000000
5 1 0 0 2 1.000000
3 1 1 1 6 1.000000
1 1 2 0 3 1.000000
3 1 2 0 3 1.000000
2 0 0 1 6 1.000000
3 2 1 0 5 1.000000
2 0 2 0 1 1.000000
0 0 1 0 0 1.000000
```

```
1 1 0 0 2 1.000000
5 0 0 1 6 1.000000
5 0 1 0 0 1.000000
1 2 1 0 5 1.000000
0 2 1 0 5 1.000000
4 2 1 0 5 1.000000
1 0 0 1 6 1.000000
4 0 0 1 6 1.000000
2 1 2 0 3 1.000000
4 1 0 0 2 1.000000
0 1 1 95.000000     //Rewards defined as:
0 1 2 0.000000      // Initial State
1 0 0 95.000000     // Imperfect Information Player action
4 2 2 95.000000     // Perfect Information Player action
2 0 2 0.000000      // Reward given to Imperfect Information player
2 1 1 95.000000
0 0 2 0.000000
1 1 0 0.000000
2 2 0 0.000000
5 1 0 0.000000
3 2 1 0.000000
3 2 2 95.000000
2 1 2 0.000000
0 2 1 0.000000
4 0 2 0.000000
2 2 1 0.000000
3 1 2 0.000000
4 0 1 0.000000
1 0 1 0.000000
2 1 0 0.000000
4 2 0 0.000000
2 2 2 95.000000
0 1 0 0.000000
1 1 2 0.000000
0 2 2 95.000000
5 0 2 0.000000
4 2 1 0.000000
3 0 2 0.000000
4 1 2 0.000000
1 2 0 0.000000
3 1 0 0.000000
4 0 0 95.000000
5 2 1 0.000000
5 2 2 95.000000
1 2 1 0.000000
3 2 0 0.000000
```

```
5 1 2 0.000000
0 0 0 95.000000
1 1 1 95.000000
5 2 0 0.000000
1 0 2 0.000000
3 1 1 95.000000
4 1 1 95.000000
6 3 3 0.000000
3 0 1 0.000000
2 0 0 95.000000
5 1 1 95.000000
5 0 1 0.000000
5 0 0 95.000000
3 0 0 95.000000
1 2 2 95.000000
0 0 1 0.000000
0 2 0 0.000000
2 0 1 0.000000
4 1 0 0.000000
0 1.0000 0.0000
```

Structure of the game definition is as follows:

1. 1 line contains information about the game as the number of states $S$, number of partitions $Pa$, number of imperfect information player actions $A_1$, number of perfect information player actions $A_2$, number of observation $O$, number of transitions $P$, number of rewards $R$, discount factor $\gamma$.
2. $S$ lines contain the definition of states. The first value is the name of the state and the second one is a partition of the state.
3. $A_1$ lines contain names of imperfect information player actions.
4. $A_2$ lines contain names of perfect information player actions.
5. $O$ lines contain names of possible observations.
6. $S$ lines contain possible perfect information player action in the state. The first line with this information corresponds to the first defined state etc.
7. $Pa$ lines contain possible imperfect information player action in the partition. The first line with this information corresponds to partition with index 0 etc.

8. $P$ lines contain all possible transitions, which is in the format: the initial state, the action played by imperfect information player, the action played by perfect information player, the observation given to imperfect information player, the resulting state and the probability of this transition for given state and actions.
9. $R$ lines contain all rewards, which are in the format: the initial state, the action played by imperfect information player, the action played by perfect information player, the reward given to imperfect information player
10. Last row contains initial belief, the first value is the initial partition, and other values are probability distribution over states in this partition.

Order within the file is important and always remains the same.

# Appendix B

# Output from HSVI

```
coords 0 0                           //Strategy Node - Partition
policy 0.333333 0.333333 0.333333    //Probability Distribution over actions
ao 0 0                               //Action - Observation
obs 0 1                              //Resulting Strategy Node - Probability
ao 0 1
obs 3 1
ao 1 0
obs 1 1
ao 1 1
obs 3 1
ao 2 0
obs 2 1
ao 2 1
obs 3 1
end                                  //Ending definition of Strategy Node
coords 1 1
policy 0.333333 0.333333 0.333333
ao 0 0
obs 0 1
ao 0 1
obs 3 1
ao 1 0
obs 1 1
ao 1 1
obs 3 1
ao 2 0
obs 2 1
ao 2 1
```

```
obs 3 1
end
coords 2 2
policy 0.333333 0.333333 0.333333
ao 0 0
obs 0 1
ao 0 1
obs 3 1
ao 1 0
obs 1 1
ao 1 1
obs 3 1
ao 2 0
obs 2 1
ao 2 1
obs 3 1
end
coords 3 3
policy 1
ao 3 1
obs 3 1
end
init 0                              //Initial Strategy Node
```

Structure of the file is that if the line starts with the word "coords" it describes a strategy node. The first number is the unique index of this strategy node and second is the index of a partition. Lines with the word "policy" describes probability distribution over possible actions in strategy node. When the line starts with the word "ao", the first number is the index of imperfect information action from game and the second number is the index of observation. Lines starting with obs describes observation from previous ao line. The first number is the index of the strategy node which may result after action-observation pair from the previous line and the second number is the probability of this strategy node. In this example, all of these probabilities are 1, but in larger games, the same action-observation pair may result in more different strategy nodes and then probability has to be lower than 1. The last line in every output starts with word init, is an index of strategy node, which is initial in the game.

# Appendix C

# Bibliography

[ARS$^+$03] Micah Adler, Harald Räcke, Naveen Sivadasan, Christin Sohler, and Berthold Vöcking, *Randomized pursuit-evasion in graphs*, Combinatorics, Probability and Computing **12** (2003), no. 3, 225–244.

[Bar22] E.N. Barron, *Game theory*, 2. ed., John Wiley & Sons, Inc, Hoboken, NJ, USA, 2013-04-22.

[BBB$^+$13] Scott Backhaus, Russell Bent, James Bono, Ritchie Lee, Brendan Tracey, David Wolpert, Dongping Xie, and Yildiray Yildiz, *Cyber-physical security*, IEEE Transactions on Smart Grid **4** (2013), no. 4, 2320–2327.

[ESB$^+$17] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer, *Pomdps.jl: A framework for sequential decision making under uncertainty*, Journal of Machine Learning Research **2017** (2017), no. 18, 1–5.

[Gut97] Joel M. Guttman, *The explanatory power of game theory in international politics*, Economics and Politics **9** (1997), no. 1, 71–85.

[HBP17] Karel Horák, Branislav Bošanskỳ, and Michal Pěchouček, *Heuristic search value iteration for one-sided partially observable stochastic games*, Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[KS] Levente Kocsis and Csaba Szepesvári, *Bandit based monte-carlo planning*, Machine Learning: ECML 2006, Springer Berlin Heidelberg, Berlin, Heidelberg.

[KSJ15] Alihan Kabalak, Elena Smirnova, and Jürgen Jost, *Non-cooperative game theory in biology and cooperative reasoning in humans*, Theory in Biosciences **134** (2015), no. 1-2, 17–46.

[LX13] Xiannuan Liang and Yang Xiao, *Game theory for network security*, IEEE Communications Surveys & Tutorials **15** (2013), no. 1, 472–486.

[RMY$^+$05] Sui Ruan, Candra Meirina, Feili Yu, Krishna Pattipati, and Robert Popp, *Patrolling in a stochastic environment*, 28.

[RN16] Stuart J. Russell and Peter Norvig, *Artificial intelligence*, third edition ed., Pearson, Boston, [2016].

[Sam16] Larry Samuelson, *Game theory in economics and beyond*, Journal of Economic Perspectives **30** (2016), no. 4, 107–130.

[SS73] Richard D Smallwood and Edward J Sondik, *The optimal control of partially observable markov processes over a finite horizon*, Operations research **21** (1973), no. 5, 1071–1088.

[SS04] Trey Smith and Reid Simmons, *Heuristic search value iteration for pomdps*, Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, AUAI Press, Arlington, Virginia, USA, 1. ed., 2004, pp. 520–528.

[SV10] David Silver and Joel Veness, *Monte-carlo planning in large pomdps*, Advances in Neural Information Processing Systems 23 (Vancouver, British Columbia, Canada), Curran Associates Inc., 2010, pp. 1–9.

# BACHELOR‘S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Kubíček Ondřej** | Personal ID number: | **474745** |
| Faculty / Institute: | **Faculty of Electrical Engineering** | | |
| Department / Institute: | **Department of Control Engineering** | | |
| Study program: | **Cybernetics and Robotics** | | |

## II. Bachelor’s thesis details

Bachelor’s thesis title in English:

**Best response computation for partially observable stochastic games**

Bachelor’s thesis title in Czech:

**Algoritmy výpočtu nejlepší odpovědi ve stochastických hrách s neúplnou informací**

Guidelines:

Solving Partially Observable Stochastic Games (POSGs) is intractable in general and thus the research is focused on subclasses of POSGs, such as One-Sided POSGs where only one player has imperfect information and the opponent has full information. The first recently developed algorithm PG-HSVI [1] for this class of games has limited scalability and there is a need for more scalable algorithms that do not have convergence guarantees. Comparison of such computed strategies can be done using a scalable best response algorithm that is currently missing. The goal of the student is to:
1) study One-Sided POSGs and their algorithms
2) formulate the best response evaluation of strategies as problem of solving a Partially Observable Markov Decision Process (POMDPs)
3) select an appropriate solver for solving the POMDP
4) experimentally evaluate the scalability, identify bottlenecks, and propose possible improvements to formulation/algorithm in order to improve the scalability.

Bibliography / sources:

[1] Horák, K., Bošanský, B., & Pěchouček, M. (2017). Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In AAAI (pp. 558-564).
[2] Silver D, Veness J. Monte-Carlo planning in large POMDPs. InAdvances in neural information processing systems 2010 (pp. 2164-2172).

Name and workplace of bachelor’s thesis supervisor:

**Mgr. Branislav Bošanský, Ph.D., Artificial Intelligence Center, FEE**

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **10.01.2020** Deadline for bachelor thesis submission: **14.08.2020**

Assignment valid until:
**by the end of winter semester 2021/2022**

| | | |
|---|---|---|
| Mgr. Branislav Bošanský, Ph.D. | prof. Ing. Michael Šebek, DrSc. | prof. Mgr. Petr Páta, Ph.D. |
| Supervisor’s signature | Head of department’s signature | Dean’s signature |

## III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

| | |
|---|---|
| . | |
| Date of assignment receipt | Student’s signature |