Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Computing Correlated Equilibria in Partially Observable Stochastic Games

Bc. Petr Tomášek

Supervisor: Mgr. Branislav Bošanský, Ph.D.

Study Programme: Open Informatics

Specialisation: Artificial Intelligence

May 24, 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tomášek** Jméno: **Petr** Osobní číslo: **425064**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Výpočet korelovaného ekvilibria v částečně pozorovatelných stochastických hrách

Název diplomové práce anglicky:

Computing Correlated Equilibria in Partially Observable Stochastic Games

Pokyny pro vypracování:

Partially Observable Stochastic Games are in general difficult to solve due to the problem of nested beliefs. Solving competitive settings (e.g., zero-sum games) is thus limited to settings where the imperfect information is restricted (for example games where only one player has imperfect information). However, computing Correlated equilibrium (CE) is computationally easier for many classes of games and can be very useful for many scenarios where the cooperation of rational agents needs to be achieved. The goal of the student is thus to (1) analyze existing algorithms for computing CE for stochastic games (with full or partial information), (2) design and implement an algorithm for computing (approximate) CE in stochastic games with imperfect information, (3) experimentally analyze the scalability of the new algorithm.

Seznam doporučené literatury:

[1] Murray, C., and Gordon, G. "Finding correlated equilibria in general sum stochastic games." Carnegie Mellon University, School of Computer Science, Machine Learning Department. 2007
[2] MacDermed, L., Narayan, K.S., Isbell Jr, C.L. and Weiss, L., "Quick Polytope Approximation of All Correlated Equilibria in Stochastic Games." In AAAI. 2011
[3] Horak, K., Bosansky, B., and Pechoucek, M. "Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games." In AAAI. 2017

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Branislav Bošanský, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Mgr. Branislav Bošanský, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Aknowledgements

I would like to thank my advisor, Mgr. Branislav Bošanský, Ph.D., for his guidance and being helpful whenever it was necessary. I would also like to thank my family and friends for support throughout my whole life.

Declaration

I declare that presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on May 24, 2018 .....

Abstract

In the real world, we have to deal with situations requiring cooperation of participating agents keeping their rationality. These problems are addressed by the game theoretical concept of correlated equilibrium. There are some works focusing on the problem of computing correlated equilibria in stochastic games. So far there is no algorithm capable of computing correlated equilibria in general partially observable stochastic games.

In this work, we propose the first algorithm for approximating correlated equilibria in partially observable stochastic games that iteratively solves these games using gradually expanding generated subset of belifestates. Even though the algorithm has no optimality guarantees, we show that it is capable to compute reasonable solutions.

Abstrakt

V reálném světě se musíme vypořádávat se situacemi vyžadujícími kooperaci zúčastněných agentů při zachování jejich racionality. Takovéto problémy odpovídají herně teoretickému konceptu korelovaného ekvilibria. Existuje několik prací, které se zabývají výpočtem korelovaného equilibria v stochastických hrách. V současnosti však neexistuje žádný algoritmus, který by byl schopen počítat korelované ekvilibrium pro obecné částečně pozorovatelné stochastické hry.

V této práci představujeme první algoritmus pro aproximaci korelovaného ekvilibria v částečně pozorovatelných stochastických hrách, který řeší tyto hry iterativně pomocí postupného zvětšování vygenerované podmnožiny belief stavů. Přestože náš algoritmus nemá žádné garance optimality, ukazujeme, že je schopen nalézt přijatelná řešení.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

There are many real-world situations involving group of players called multi-agent problems. For many of these problems, agents can get better outcomes when they cooperate. However, they will not agree with just any cooperative strategy since they are rational and they want primarily to maximize their own outcome. Thus they will cooperate with other agents only if this is the best option for them. In other words, an agent will participate in cooperative strategy only if his action is the best response to actions of other agents.

These conditions also hold for problems addressed by game theory and the best fitting class of games are stochastic games. There are several game-theoretical solution concepts for such problems. When we want agents to coordinate but not to make alliances we need to choose an action for each agent such that it is the best response to actions taken by other players. This is exactly what correlated equilibrium does. The main idea of correlated equilibrium solution concept is that we have some correlation device that knows the real state of the game and provides a recommendation to each player about the action he should take. These recommendations are chosen in such way that any player cannot do any better by deviating from his recommendation, which means the recommended action is a best response.

There exist several works dealing with computing correlated equilibria in stochastic games. There are also some works addressing this computation in partially observable environments. However, solutions for partially observable environments do not assume general partially observable stochastic games but only special subsets. Probably computationally easiest to solve and the best-mapped subset of partially observable stochastic games are partially observable stochastic games with common payoff also know as decentralized partially observable Markov decision processes.

In this work, we address the problem of computing correlated equilibria in general partially observable stochastic games. To the best of our knowledge, there does not exist such algorithm yet. We provide the first algorithm for approximating correlated equilibria in general partially observable stochastic games and its experimental evaluation.

One of the most interesting types of games where we want a group of players to act in a coordinated manner is pursuit-evasion games. In these games, a group of pursuers tries to catch an evader. Since such games correspond to important real-world problems, we decided to choose our running example from this type of problems.

In the game, we chose we have highway patrols trying to catch a bank robber on the run. The situation can be that highway patrols are resting at their base when they receive information about a bank robbery. They do not hesitate and immediately start the hunt on the robber. The robber wants to get away as fast as possible and tries to escape using a network of high-speed roads. Unfortunately for patrols all vehicles using these high-speed roads move really fast and therefore they can catch and spot the robber only at specific places such as crossings and exits and not on the road itself. Since patrols want to catch as fast as possible they do not have time for long communications thus each patrol has to act without knowing where exactly the other patrols are. We can think of correlation device as police central with full information about current state. The police central cannot control the patrols directly but can give them some advice what should they do. The game does not have a strict horizon and when patrols catch the robber the game starts from the beginning.

In this work we provide an algorithm that computes strategies that can be used in described scenarios. The work itself is structured as follows. First of all, we provide the technical background needed for understanding our work in Chapter 2. We define different classes of games (Section 2.1) namely Markov decision processes (Subsection 2.1.1), partially observable Markov decision processes (Subsection 2.1.2), stochastic games (Subsection 2.1.3), partially observable stochastic games (Subsection 2.1.4), Bayesian games (Subsection 2.1.5), decentralized partially observable Markov decision processes (Subsection 2.1.6), value iteration baseline algorithm (Subsection 2.2.1) and solution concept of correlated equilibrium (Subsection 2.3.1). In Chapter 2 we also review current literature focusing on solving stochastic games (Subsection 2.4.1) and partially observable stochastic games (Subsection 2.4.2). In Chapter 3 we provide our approximate algorithm for computing correlated equilibria in partially observable stochastic games. Results obtained from the experimental evaluation are presented in Chapter 4. And finally, in Chapter 5 we summarize the results of our work and outline future work we want to make on this topic.

Chapter 2

Technical Background

In this chapter, we provide the technical background of our work. In separate sections, we describe Markov decision processes, partially observable Markov decision processes, stochastic games and partially observable stochastic games and show how they work on examples. These definitions are from my Bachelor's project [21]. Then we briefly describe Bayesian games and decentralized partially observable Markov decision processes based on the common definitions. And finally, we provide concepts of correlated equilibria ([15]) and value iteration. At the end of this chapter, we also provide a review of current works focusing on solving stochastic games and partially observable stochastic games.

2.1 Classes of Games

2.1.1 Markov Decision Processes

Markov decision processes (MDPs) provide a mathematical framework for modelling decision-making in situations where outcomes are partly random and partly under the control of a decision maker (agent). The goal is to choose an optimal action in every state to increase the expected reward. MDPs are useful for studying a wide range of optimization problems, solved via dynamic programming and reinforcement learning. They are used in a wide area of disciplines, including robotics, automated control, economics, and manufacturing.

More precisely an MDP is a discrete time stochastic control process and can be viewed as an extension of Markov chains with a set of actions and state-based reward function. An MDP is described by 5-tuple (S, A, T, R, γ), where:

- $\boldsymbol{S}$ is a finite set of possible states
- $\boldsymbol{A}$ is a finite set of actions
- $\boldsymbol{T}$ is a transition function, T: S x $A \rightarrow \Delta(S)$
- $\boldsymbol{R}$ is a reward function, R: S x $A \rightarrow \mathbb{R}$
- $\boldsymbol{\gamma}$ is the discount factor representing the difference between weights of actual and future rewards, $\gamma \in [0, 1]$

2.1.1.1 States and Actions

A state represents the environment at a particular point in time. Since the state space S is finite and the process is stochastic, a particular state can be viewed as a random variable s_k whose domain is the state space. Each state contains a finite set of agent's possible actions A. The set A can be the same for all states but it may differ. Moreover for process to be Markovian, a state has to contain enough information for next state prediction. In other words the history is irrelevant to predict the future.

2.1.1.2 Transition Function

In every state, the agent chooses an action from action set and executes it. This affects the state transition probabilities according to transition function T. This function gives a probability distribution over possible future states for each state and action. That means for each $s_k, s_{k+1} \in S$ and $a \in A$, the function T determines the probability of ending up in state s_{k+1} after executing action a in state s_k.

2.1.1.3 Reward Function

Reward function R assigns a numeric reward to each pair (s_k, a). Thus for each $s_k \in S$ and $a \in A$, the function R gives a reward obtained by the agent after executing action a in state s_k.

2.1.1.4 Strategy

A strategy in MDP is a direct mapping from a state to a distribution over the possible actions set A.

$$\pi : S \to \Delta(A) \tag{2.1}$$

2.1.1.5 Example

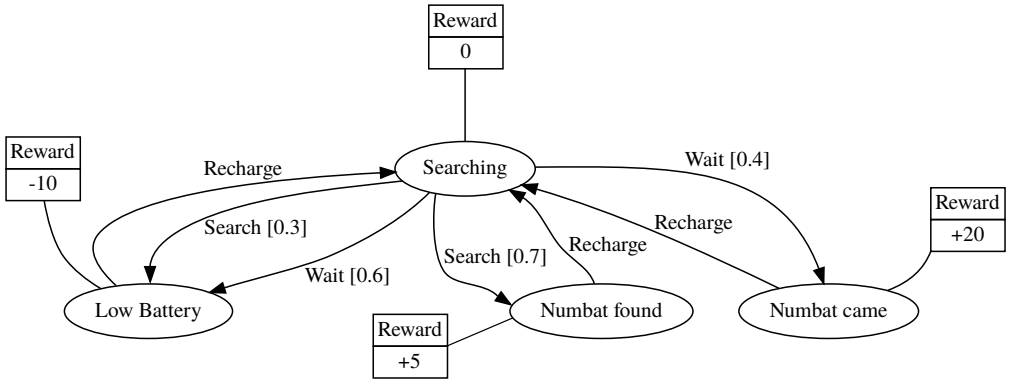


Figure 2.1: Example of Markov decision process

To show how MDPs work we assume the situation of a robot photographing numbats in their natural habitat (Figure 2.1). For higher clarity of example scheme, we included the rewards into the states.

Australian scientists want to study numbats in their natural habitat. Therefore they placed the robot in the Dryandra Woodland in Western Australia and gave him very simple task: take photos of numbats in the best quality possible.

The robot has two possible ways of encountering numbats, wait for them near the base or actively search for them. If he actively searches for numbats he has quite high chance of encounter. But numbats are easily scared, thus taken photos do not have the highest quality. If he chooses to wait for numbats near his base he can take photos in high quality but there is also high chance of getting low on battery.

When robot finds a numbat he takes a photo and obtains a positive reward according to the photo quality, then he recharges his battery and continues his task. If he gets low on battery during searching or waiting for numbat he obtains a negative reward as a representation of wasted time.

2.1.2 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process (POMDP) is a generalization of a Markov Decision Process (MDP) to situations in which an agent cannot directly observe the states. This generalization increases the complexity of problems that can be represented. It is general enough to model real-world sequential decision processes. The goal is to find the optimal action for each possible belief state which maximizes the expected utility. Applications include robot navigation problems, machine maintenance, and planning under uncertainty in general.

Formally a POMDP is comprised of an underlying MDP, extended with an observation set O and observation function Ω. In other words POMDP is described by 7-tuple $(S, A, O, T, \Omega, R, \gamma)$, where:

- $\boldsymbol{S}$ is a finite set of possible states
- $\boldsymbol{A}$ is a finite set of actions
- $\boldsymbol{O}$ is a finite set of observations
- $\boldsymbol{T}$ is a transition function, T: S x $A \to \Delta(S)$
- $\boldsymbol{\Omega}$ is a observation function, Ω: S x $A \to \Delta(O)$
- $\boldsymbol{R}$ is a reward function, R: S x $A \to \mathbb{R}$
- $\boldsymbol{\gamma}$ is the discount factor representing the difference between weights of actual and future rewards, $\gamma \in [0, 1]$

2.1.2.1 Observations and Observation Function

In MDPs, the agent has full knowledge of the system state, thus $O \equiv S$. In POMDPs observations are only probabilistically dependent on the underlying environment state which makes determining the state the agent is in problematic, because the same observation can be observed in different states. It is assumed that agent has perfect recall of actions he took and observations he observed.

An observation function Ω specifies the relations between states and observations. This is made by assigning a probability to triplets (s_{k+1}, a, o_{k+1}). Thus the function Ω returns the probability of observing observation o_{k+1} after executing action a and ending up in state s_{k+1}.

2.1.2.2 Strategy

A strategy in POMDP is a mapping from observation history h to a distribution over the possible actions set A.

$$\pi : h \to \Delta(A) \tag{2.2}$$

2.1.2.3 Example

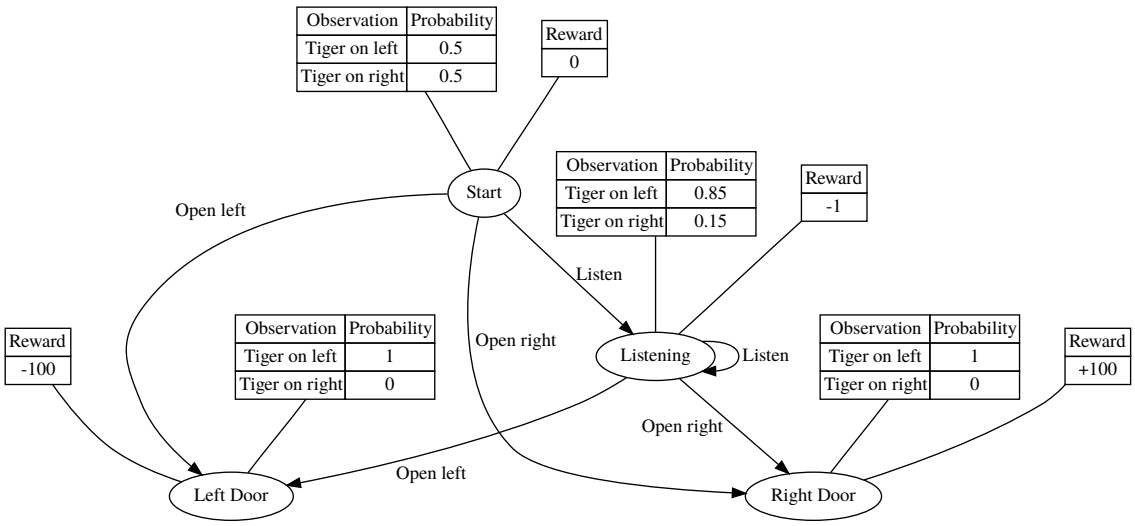


Figure 2.2: Example of partially observable Markov decision process

To show how POMDPs work we assume the situation of an agent closed in a room with two possible exits and tiger behind one of them (Figure 2.2).

James Bond, a British Secrete Service agent code number 007, was captured and placed in cell by sir Hugo Drax. Drax likes to play games with his prisoners, thus he left possibility to escape. James' cell has two possible exits, door on left-hand side and other door on right-hand side, but only one of them can set James free. Because Drax is an insidious villain he placed a tiger behind the door on the left. Unfortunately James doesn't know that. Could he escape and stop Drax or will he be eaten?

Being inside the cell James has three possible actions to take, open the door on the left, open the door on the right and listen. Choosing to open the door results in direct reward according to surviving or dying. If James chooses to listen he wastes some time and gains negative reward, on the other hand he obtains observation that can make his chances to survive and stop Drax higher.

Imagine James just wakes up and realizes the situation he is in. Thus James is in the "Start" state and needs to make his first decision. His belief of the tiger being hidden behind door on the left is the same as tiger behind door on the right. Directly choosing one door can be the quickest way to get out but also the quickest one to die. Therefore James chooses to spend a little bit longer in cell and listen to improve his chances ending up in "Listening" state and obtains negative reward of -1 for wasted time. He receives observation that the tiger is on the side he is really hidden with probability 0.85 or tiger being on the other side with probability 0.15. This observation changes his belief about the environment and he decides again whether he is sure enough to pick one door to open or he still needs to listen. The process repeats until he picks one door. If he finally chooses the wrong door on the left he dies and gets negative reward -100 ending up in "Left Door" state. If he chooses the right door he ends up in "Right Door" state receiving positive reward of +100 for surviving and stopping Drax.

2.1.3 Stochastic Games

A fully observable stochastic game is a dynamic game with probabilistic transitions played by one or more players. The particular game state at any time depends probabilistically on the previous state and the actions taken by players in that state. Stochastic games are widely used for modelling and analysing discrete systems operating in an adversarial environment such as in economics, evolutionary biology or computer networks.

Stochastic games generalize both Markov Decision Processes (MDPs) and repeated games. Therefore, there are many key ingredients in common. A stochastic game is described by:

- a finite set of players $\boldsymbol{I}$
- a finite set of possible states $\boldsymbol{S}$
- a finite action set $\boldsymbol{A_i}$ for each player i
- transition function $\boldsymbol{\psi}$: S x A_1 x A_2 x ... x $A_{|I|} \rightarrow \Delta(S)$
- reward function $\boldsymbol{\rho}$: S x A_1 x A_2 x ... x $A_{|I|} \rightarrow \mathbb{R}^{|I|}$

2.1.3.1 States and Actions

A state represents the current state of the game. Each state contains a finite set of possible actions for each player, which may differ across states. Moreover each state also contains enough information for making a decision. That means the history is irrelevant for decision and it is enough to use a direct mapping from a state to a distribution over possible actions as strategy. This fact was proved by Shapley in his seminal paper [19] where he proved that each player has an optimal strategy that depends only on the current state and not on past history.

2.1.3.2 Transition Function

In every state, players independently choose an action from their action sets and execute it. This affects the state transition probabilities according to transition function ψ. This function returns a probability distribution over possible future states for each state and joint action. In two-player case for each $s_k, s_{k+1} \in S$, $a_1 \in A_1$ and $a_2 \in A_2$, the function ψ determines the probability of ending up in state s_{k+1} after executing joint action (a_1, a_2) in state s_k.

2.1.3.3 Reward Function

Reward function ρ assigns a vector from $\mathbb{R}^{|I|}$, containing numeric reward for every agent, to each pair s_k and joint action a. In two-player case for each $s_k \in S$, $a_1 \in A_1$ and $a_2 \in A_2$, the function ρ returns vector (r_1, r_2) where r_1 is reward obtained by the agent 1 and r_2 is reward received by agent 2 after executing joint action (a_1, a_2) in state s_k.

2.1.3.4 Strategy

As we already stated and Shapley proved in his seminal paper [19] it is sufficient to use a direct mapping from a state to a distribution over the possible actions set A_i as a strategy in stochastic game.

$$\sigma : S \to \Delta(A_i) \tag{2.3}$$

2.1.3.5 Example

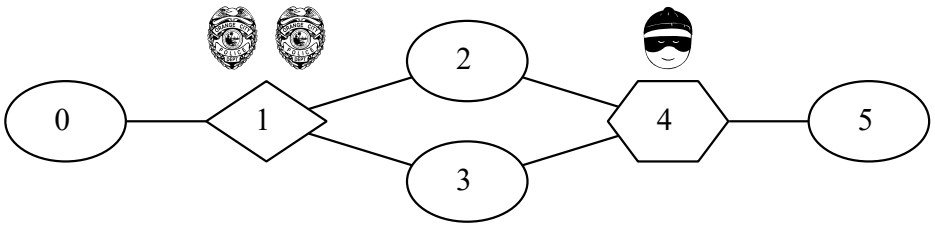


Figure 2.3: Example of stochastic game

To show how stochastic games with perfect information work we use the game shown in Figure 2.3. This game is an instance of our running example used for experiments and represent the situation when a group of pursuers is trying to catch an evader. For demonstration, lets assume that there are two pursuers in team starting at the diamond node and an evader starting at hexagon node.

A stochastic game is played as follows. Players independently and simultaneously choose and execute their actions for the game state they are in. Then stage rewards and transition

are calculated. Information about actions, rewards and new stage is told to players and game proceeds to next stage and the procedure repeats for the rest of the game.

In our case players (pursuers) start in the diamond node 1 and their possible actions are moving to node 0, node 2 or node 3. The evader starts in hexagon node 4 and uniformly randomly moves to node 2, node 3 or node 5. Players choose and take their actions. For example, player 1 moves to the node 2 and player 2 to the node 3. This can result in catching evader at node 2 by player 1 with probability 0.33, catching evader at node 3 by player 2 with probability 0.33 or the evader escaping to the node 5 also with probability 0.33. Assume that according to the probabilities the evader randomly chose to move to the node 5. Thus the players are told that game is now in the state where player 1 is at node 2, player 2 at node 3 and the evader is hiding at node 5 and players get rewards according to their reward functions. Then players choose and play their actions for the current state of the game, transition probabilities and rewards are calculated and the game proceeds to the next stage.

2.1.4 Partially Observable Stochastic Games

Formally a partially observable stochastic game (POSG) is comprised of an underlying stochastic game, extended with set of observations O_i for each player i. In other words a stochastic game with imperfect information is described by:

- a finite set of players $\boldsymbol{I}$
- a finite set of possible states $\boldsymbol{S}$
- the initial state distribution $\boldsymbol{b^0} \in \Delta(S)$
- a finite action set $\boldsymbol{A_i}$ for each player i
- a finite set of observations $\boldsymbol{O_i}$ for each player i
- transition function $\boldsymbol{\psi}$: S x A_1 x A_2 x ... x $A_{|I|} \rightarrow \Delta(O_1, O_2, ..., O_{|I|}, S)$
- reward function $\boldsymbol{\rho}$: S x A_1 x A_2 x ... x $A_{|I|} \rightarrow \mathbb{R}^{|I|}$

2.1.4.1 States and Actions

A state represents the environment at a particular point in time. Since the state space S is finite and the process is stochastic, a particular state can be viewed as a random variable s_k whose domain is the state space. Each state contains a finite set of agent's possible actions A. The set A can be the same for all states but it may differ. Moreover for process to be Markovian, a state has to contain enough information for next state prediction. In other words the history is irrelevant to predict the future.

2.1.4.2 Transition Function

In every state, the agent chooses an action from action set and executes it. This affects the state transition probabilities according to transition function T. This function gives a probability distribution over possible future states for each state and action. That means for each $s_k, s_{k+1} \in S$ and $a \in A$, the function T determines the probability of ending up in state s_{k+1} after executing action a in state s_k.

2.1.4.3 Reward Function

Reward function R assigns a numeric reward to each pair (s_k, a). Thus for each $s_k \in S$ and $a \in A$, the function R gives a reward obtained by the agent after executing action a in state s_k.

2.1.4.4 Observation Sets and Transition Function

When playing a partially observable stochastic game only the initial state distribution b^0 is told to the players. Further orientation in the game is possible only through signals provided to players by environment based on probabilities calculated by transition function, which works quite similar as in fully observable stochastic game. Therefore players cannot determine current state of the game for sure. However game state is not the only thing they have to consider. Since current state of the game is unknown to all players each player has his own belief about current state of the game.

Decision of each player is affected by his belief. The reward of player i is calculated based on played join action which means it depends not only on belief of player i but also on the actions taken by other players. However actions of other players depends on their beliefs thus each player should have belief also about beliefs of other players. Unfortunately also these beliefs affects played actions and rewards. Therefore each player needs belief about these beliefs too. We can easily see that nesting of beliefs can be infinite.

In every state, each player independently chooses an action from his action set and executes it. This affects not only the state transition probabilities but also the information obtained by other players according to transition function ψ. This function gives a probability distribution over possible future states and signals received by players for each state and joint action. Thus in two-player case for each $s_k, s_{k+1} \in S$, $a_1 \in A_1$, $a_2 \in A_2$, $o_1 \in O_1$ and $o_2 \in O_2$, the function ψ determines the probability of ending up in state s_{k+1} when player 1 observed signal o_1 and player 2 signal o_2 after executing joint action (a_1, a_2) in state s_k.

2.1.4.5 Strategy

A strategy in partially observable stochastic game is a mapping from belief b_i to a distribution over the possible actions set A_i.

$$\sigma : b_i \rightarrow \Delta(A_i) \tag{2.4}$$

2.1.4.6 Example

To show how partially observable stochastic games work we use just the same game we used for explaining stochastic games (shown in Figure 2.3). The only difference will be that we assume that the game is not fully observable.

A partially observable stochastic game is played as follows. At each timestep of game players independently and simultaneously choose and execute their actions from their action sets. Then stage rewards, transitions and observations are calculated. Only observations are provided to players and game proceeds to next stage. This stage is unknown to the players. Thus every time the procedure repeats they need to take into account observations they received. The procedure is repeated for the rest of the game.

Lets assume the same scenario we used for stochastic games. Players (pursuers) start in the diamond node 1 and their possible actions are moving to node 0, node 2 or node 3. The evader starts in hexagon node 4 and uniformly randomly moves to node 2, node 3 or node 5. Players choose and take their actions. For example player 1 moves to the node 2 and player 2 to the node 3. This can again result in catching evader at node 2 by player 1 with probability 0.33, catching evader at node 3 by player 2 with probability 0.33 or the evader escaping to the node 5 also with probability 0.33. Then players get their observations. Since the game is just partially observable the observation does not automatically reveal the real state the game is in and can be only used by a player to make his belief about possible states the game can be in. After establishing their beliefs players need to choose their next actions.

If players immediately catch the evader the game proceeds to initial state. If the evader is not caught it means that the evader moved to the node 5. In such situation the belief of player 1 will be that player 2 can be in node 0 or node 3 with equal probability of 0.5 and the evader can be in node 3 or node 5 with the probability of 0.5.

The belief of player 2 will be almost similar. Player 2 will believe that player 1 is in node 0 or node 2 each with probability 0.5 and the evader is in node 2 or node 5 again with equal probability of 0.5.

Thus when players choose their next actions they need to take these possibilities into account and try to choose an action which seems to be the best one over all of them.

2.1.5 Bayesian Games

Bayesian games model single state problems in which each agent has private information about something relevant to the decision making process. Formally a Bayesian game is described by:

- a finite set of players $\boldsymbol{I}$
- a finite set of actions $\boldsymbol{A_i}$ for each player i
- a type space of player $\boldsymbol{\Theta_i}$ for each player i
- a probability distribution $\boldsymbol{p}$ over type profile space $\Theta = \{\Theta_1, \Theta_2, ..., \Theta_{|I|}\}$ used to assign types to agents
- a reward function $\boldsymbol{R}$: A_1 x A_2 x ... x $A_{|I|} \rightarrow \mathbb{R}$

It is assumed that probability distribution p is common knowledge.

2.1.5.1 Strategy

By definition an agent's strategy in Bayesian game must assign an action for each of its possible types.

$$\pi : \Theta \to \Delta(A) \tag{2.5}$$

2.1.6 Decentralized Partially Observable Markov Processes

Decentralized partially observable Markov processes (DecPOMDPs) are an extension of the standard POMDPs (by adding more players). They are also known as common-payoff POSGs. As the alternative name common-payoff POSGs indicates they are a special case of POSGs. Since that they are described and work the exact same way as standard POSGs. The only difference is that all players share their payoffs (all agents have identical reward function).

2.2 Baseline Algorithms

2.2.1 Value Iteration

Value iteration is a popular and common method for computing an optimal policy and its value. At first, it was designed for solving single agent problems like MDPs. In each iteration the algorithm evaluates all states. During backup step it refines an estimate of either Q^* or V^* depending on whether the q-function or value function is chosen. In our work we used the variant with value function, therefore remaining description will assume this variant.

Let V_k be the value function assuming there are k stages left to play. Such function can be easily defined recursively. Value iteration starts with an arbitrary function V_0 and performs maximization over actions. Following equation is used to get the function for $k+1$ stages left from the function for k stages:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s')) \qquad \forall k \geq 0 \tag{2.6}$$

Algorithm 1 shows the pseudocode of baseline value iteration algorithm. This algorithm converges no matter what is the initial value function V_0. However, the convergence can be faster when using V_0 which approximates V^*.

We can also modify the single agent variant of this algorithm into multi-agent one. This involves replacing the maximization over actions (in Algorithm 1 on row 20 and row 24) with an equilibrium computation. This modified version achieve a few successes particularly in the zero-sum case where strong convergence guarantees have been achieved [7]. Unfortunately in general-sum cases for both Nash and correlated equilibria we can encounter phenomenon known as cyclic equilibria [9].

The cyclic equilibria appears when we have states that form a cycle and are connected in such manner that these states are forced to periodically change their equilibrium solutions

Algorithm 1 Value Iteration

procedure ValueIteration($S, A, P, R, \Theta, \gamma$)
  Input:
    S is the set of all states
    A is the set of all actions
    P is state transition function specifying $P(s'|s, a)$
    R is a reward function $R(s, a, s')$
    Θ is a threshold $\Theta > 0$
    γ is a discount factor $0 < \gamma < 1$
  Output:
    $\boldsymbol{\pi}[\boldsymbol{S}]$ approximately optimal policy
    $\boldsymbol{V}[\boldsymbol{S}]$ value function
  Local:
    real array $\boldsymbol{V_k}[\boldsymbol{S}]$ is a sequence of value functions
    actions array $\boldsymbol{\pi}[\boldsymbol{S}]$
  assign $V_0[S]$ arbitrarily
  $k \leftarrow 0$
  repeat
    $k \leftarrow k + 1$
    for $s \in S$ **do**
      $V_k[s] = \max_a \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma V_{k-1}[s']))$
    end for
  until $|V_k[s] - V_{k-1}[s]| < \Theta \qquad \forall s \in S$
  for $s \in S$ **do**
    $\pi[s] = \arg\max_a \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma V_{k-1}[s']))$
  end for
  return π, V_k
end procedure

and values. Since the difference between values can be greater then termination threshold, the value iteration may run forever. The commonly used solution is to add cycle detection. To be able to detect the cycle we need to store the history of previously reached equilibria for the particular state. This can be done either for all states or just a subset of states we are concerned about. When we reach the same equilibrium in some later iteration the cycle is detected and the value iteration terminates. As a result we can use either last obtained solution or we can choose the best solution among the ones forming the cycle based on our criteria.

2.3 Solution Concepts

2.3.1 Correlated Equilibrium

Correlated equilibrium is a solution concept generalizing the well know Nash equilibrium. The high-level idea is that players are receiving recommendation what action they should

take from correlation device (e.g. traffic lights). If no player wants to deviate from the recommendation assuming the other will not deviate too, the resulting strategy is called correlated equilibrium.

Lets assume an one-shot N-player matrix game where each player p has a finite set of pure strategies A_p. The set $A = \prod_{p=1}^{n} A_p$ is called the set of strategy profiles. Strategy profile without player p is denoted as $A_{-p} = \prod_{p \neq q} A_q$. The utility function of player p at state s is a function u^p mapping set of strategy profiles A to the numerical values. A distribution x on A is a vector of nonnegative real numbers, one for each strategy profile in A, summing up to 1.

The correlated equilibrium is a distribution x on A such that for all players p and all pure strategies $i, j \in A^p$ the following statement is true. Conditioned on $p - th$ component of strategy profile drawn from x being action i, the expected utility for player p of playing action i is no smaller than that of playing action j. This can be mathematically expressed as:

$$\sum_{a \in A_{-p}} [u^p_{ia} - u^p_{ja}] x_{ia} \geq 0 \qquad \forall i, j \in A^p \tag{2.7}$$

$$\sum_{a \in A} x_a = 1 \tag{2.8}$$

$$x_a \geq 0 \qquad \forall a \in A \tag{2.9}$$

Lets assume two player chicken game showed in Figure 2.4. These players are two competitive drivers speeding from different streets to an intersection. Each of them with two strategies $S_1 = S_2 = stop, go$. The utility function is specified by the matrix. When both players stop before intersection they get an equal reward of 4. In the situation when only one of the players stops before intersection the one who did not stop gets a reward of 5 and the one who stopped gets 2. And in the situation when neither of them stops they crash and obtain reward 0.

	stop	go
stop	4, 4	2, 5
go	5, 2	0, 0

Figure 2.4: Chicken game

Now consider a correlation device (for example some kind of traffic lights) that draws one of three strategy profiles (*stop*, *stop*), (*stop*, *go*), (*go*, *stop*) with uniform probability 1/3. After drawing strategy profile the correlation device informs the players of strategy assigned to them, but not the strategy assigned to their opponent. Suppose a player is assigned *go*, he would not want to deviate supposing the other player played their assigned strategy since he will get the highest possible payoff. Suppose a player is assigned *stop*. Then the other

player has been assigned *stop* with probability 0.5 and *go* with probability 0.5. The expected utility for *go* is $0.5 * 5 + 0.5 * 0 = 2.5$ and the expected utility for *stop* is $0.5 * 4 + 0.5 * 2 = 3$. Thus player will prefer to play action *stop* which means he would not want to deviate from suggested strategy.

Since neither player has an incentive to deviate from assigned, playing strategy profiles (*stop*, *stop*), (*stop*, *go*), (*go*, *stop*) with uniform probability 1/3 is a correlated equilibrium of chicken game.

The solution concept of correlated equilibrium can be easily transformed to be used for stochastic games too. In each state of a stochastic game we solve a matrix game. Thus we can compute correlated equilibrium for the state games. The only difference is in the utility function since we do not play just a one-shot game but we also have to consider rewards from future states usually weighted by the discount factor γ.

Lets assume an N-player stochastic game with set of game states S. Each game state s can be represented as a stage matrix game. In game state s has each player p a finite set of pure strategies $A_p(s)$. The set $A(s) = \prod_{p=1}^{n} A_p(s)$ is the set of strategy profiles available in state s. Strategy profile without player p is denoted as $A_{-p}(s) = \prod_{p \neq q} A_q(s)$. Lets denote the set of states reachable from state s by strategy profile a as $S(s, a)$. The continuation utility after playing action profile a in state s is $c_a^p(s) = \sum_{s' \in S(s,a)} V_{k-1}[s'] P(s'|s, a)$. Finally the constraints needed for computing correlation equilibria in stochastic games will be for each state $s \in S$:

$$\sum_{a \in A_{-p}(s)} [[u_{ia}^p + \gamma c_{ia}^p(s)] - [u_{ja}^p + \gamma c_{ja}^p(s)]] x_{ia} \geq 0 \quad \forall i, j \in A^p(s) \tag{2.10}$$

$$\sum_{a \in A(s)} x_a = 1 \quad \forall i, j \tag{2.11}$$

$$x_a \geq 0 \qquad \forall a \in A(s) \tag{2.12}$$

These constraints are sufficient to form linear program for computing correlated equilibria. Such program will replace the maximization over actions used in value iteration algorithm (in Algorithm 1 on row 20 and row 24).

2.4 Literature Review

2.4.1 Stochastic Games

Despite the fact that stochastic games (SGs) assume fully observable environment they can be used to model many problems. Since SGs generalize MDPs many attempts to solve them tries to extend Bellman equation (used in value iteration for MDPs) to domains with multiple agents.

One of the first attempts to solve stochastic games was made by Littman [6] and focuses on two-player zero-sum games. The algorithm presented in this work (minimax-Q) is just a slight modification of Q-learning algorithm used for solving MDPs (also know as value

iteration). A stochastic game is divided into states, where each state corresponds to one matrix game. In every state the matrix game is solved using minimax theorem with the assumption that players will continue optimally (Q-function). Then the value of a state is updated according to Q-function. Littman and Szepesvári showed that minimax-Q converges to the game-theoretic optimal value [8].

The algorithm developed by Littman was than extended by Hu and Wellman to two-player general-sum games [5] considering only stationary strategies. Unlike in zero-sum games in general-sum games, players need no longer have opposite interests and each player's reward depend on other players' actions. Thus the minimax approach does not work for these situations. Because of that, they adopted Nash equilibrium solution concept instead of minimax. Which means that in every state the corresponding matrix game is solved using Nash equilibrium (therefore this algorithm is known as Nash-Q). The convergence conditions of the algorithm were clarified by Bowling [1].

Although the Nash-Q algorithm is highly general, the assumptions to guarantee its convergence are quite restrictive. Only coordination or adversarial equilibrium games with specific intermediate results of learning are guaranteed to be solved. Therefore another algorithm (friend-or-foe Q-learning - FFQ) was presented by Littman [7]. This algorithm always converges and for games with adversarial or coordination equilibria converges to the exact same values as Nash-Q. To achieve this improvement FFQ algorithm needs other players to be identified as 'friend' or 'foe'. According to other players' type appropriate learning is used. In two player scenario for 'foe' it uses minimax and for 'friend' ordinary Q-learning. The key idea of FFQ algorithm is that player's 'friends' are assumed to work together to maximize its value, while its 'foes' are working together to minimize the value. Therefore n-player FFQ can treat any game as a two-player zero-sum game with an extended action set.

Another algorithm based on negotiation was developed by Gordon and Murray [13]. This algorithm finds subgame-perfect Nash equilibria for general-sum stochastic games with cheap talk (the players observe the state of the world, discuss their plans with each other and then simultaneously select their actions). In such equilibrium every possible deviation from plan is eliminated by the threat of a suitable punishment which is believable. There may be many subgame-perfect equilibria with different payoffs (some are better for one group of players, some are better for another group). In the majority of cases it is infeasible to compute full set of subgame-perfect equilibria and even in cases where it is, there is no obvious way to select the one to be implemented. Thus agents must negotiate until they find the one acceptable by all of them. It is assumed that algorithm knows possible punishment policies for each player triggered on a deviation of the particular player from plan and players agreed beforehand on a subgame-perfect equilibrium (disagreement policy), which will be played in case the negotiation fails. In each state of the game every player computes the equilibrium which is most advantageous from its point of view. After all players have their equilibria calculated the negotiation phase starts. In first phase agents take turns on revealing their equilibria to others. On agent's turn the equilibrium is revealed or agent passes. If all agents pass consecutively the negotiation proceeds to the second phase. When a player states a policy for its equilibrium, other players verify that it is really a subgame-perfect equilibrium and calculate their value for this policy. The player who stated non-equilibrium policy loses its turn. At the end of the first phase there will be a set of revealed policies.

For each revealed policy it is calculated the gain against disagreement policy. Joint gains for policies are represented as a set of points in the space and then the convex hull of the set is created. In second phase players take turns on proposing points from the convex hull along with policies needed to achieve them. After each proposal all agents except proposer decide whether the proposition will be accepted or rejected. If everyone agrees, the proposal is implemented, agents execute their actions according to it and game proceeds to next stage. If the proposal is rejected, those agents who agreed on it are removed from decision process and their utilities are fixed to the value achievable by the rejected proposal. That means all future proposals must give to those players exactly the same utility as the proposal they are willing to accept. When the negotiation fails (we hit ϵ chance of having the current round of negotiation end) all players without fixed utility are assigned to their values for disagreement policy. The players execute the proposed policy (or disagreement policy if there is no valid proposal). And any player whom value for policy to execute is higher than their assigned value voluntarily lowers their utility to the correct level. Failure to do so results in all players reverting to disagreement policy. The subgame-perfect Nash equilibrium selected by players is guaranteed to be Pareto-optimal (any single player's value cannot be improved without lowering the value to another player) and approximately fair (it is near the so-called Nash bargaining point).

Gordon and Murray improved their algorithm [14] by removing the assumption of having set of punishment policies as an input (preparing a set of suitable punishment policies can be computationally difficult). They improved the algorithm also in sense of tractability. The previous version of the algorithm was intractable since it operates on arbitrary convex sets. By representing each convex set only using a finite number of points, which can be easily done, they achieved a tractable version of their algorithm.

Another improvement to Gordon's and Murray's algorithm was done by MacDermed and Isbell [11]. They use general-sum stochastic games with cheap talk as Gordon and Murray did and assume that past joint actions are part of the state and that each player is rational. They found out that there are three main bottlenecks in the exact algorithm. The first one is that the size of the game is exponential in the number of players (since joint actions are exponential in the number of players). The second one is that although the exact algorithm always converges it is not guaranteed to converge in finite time. And finally, the third one is that maintaining an exact representation of a feasible set can cause intractability. The first problem is unavoidable unless the game is approximated. Thus they focused mainly on the latter two problems and made two modifications to the exact algorithm. First of all they added a stopping criterion. This ensures that at each iteration at least one feasible set changes by more than user-defined threshold. The second modification is representing each feasible-set as a bounded number of points. For computing expected feasible sets and correlated equilibria of sets, they use a multiobjective linear program.

2.4.2 Partially Observable Stochastic Games

Partially observable stochastic games are very general and can be used to model a wide range of real-world problems, which make them really attractive in sense of usability. However, their biggest advantage (their generality) is at the same time their biggest problem. As we have already mentioned players do not know the current state of the game but they just

have a belief about it. Since each player has his own belief, a player has to take into account beliefs of other players when he is making the decision about what action he is going to execute. Which results in need of belief about other players belief. But it is even worse since every player has his own belief about other players beliefs he needs another belief about beliefs which players hold about each other's beliefs. As we can see such way of reasoning can continue forever. Therefore the problem of solving POSGs is very difficult.

To the best of our knowledge there is no algorithm computing optimal or reasonable policies for general POSGs and all existing algorithms focus on limited subsets of POSGs. Probably computationally the easiest to solve and the best-mapped subset of POSGs are common-payoff POSGs (which are also known as decentralized Partially Observable Markov Decision Processes - DecPOMDPs).

Most of the algorithms solving common-payoff POSGs uses the fact, that these games can be transformed into a sequence of Bayesian games. This transformation (Emery-Montemerlo et.al [3]) is non-trivial since a POSG does not naturally correspond to the sequence of Bayesian games (unlike a fully observable stochastic game does). The high-level idea of this transformation is approximating of entire POSG by constructing a smaller Bayesian game at each timestep. Each smaller game models a subset of possible observations up to that time step and finds a one-step lookahead policy for each agent contingent on that subset. To model a time slice game it is essential to represent each sub-path through POSG game tree up to the desired timestep as a single entity. Such a single sub-path corresponds to a specific set of action histories and observations for all agents in the team. When all agents know that a specific sub-path has occurred the problem becomes fully observable and the payoffs of each joint action are known with certainty, which is analogous to the utility in the Bayesian games being conditioned on specific belief profiles. Each sub-path in POSG up to time t is considered as a specific belief profile Θ^t. Profiles consist of the beliefs of each agent Θ_i^t, which represents agent's own observations and actions and can appear in multiple Θ^t. Bayesian games also need agents having common prior over the belief profile space Θ. If all agents have common knowledge of starting conditions of POSG (e.g. a probability distribution over possible start states), then Θ^{t+1} and $p(\Theta^{t+1})$ can be calculated iteratively based on Θ^t, $p(\Theta^t)$, action set, observation set and transition function. Additionally, since the solution of the game is set of policies for all agents each agent knows its own policy for the next step and the policies of its teammates as well. This information can be then used to update the belief profile space too. Because of the fact that the set of possible histories can become very large, it is advised to use only beliefs with a prior probability higher than some threshold and renormalize $p(\Theta)$. Finally, there has to be a utility function that represents the payoffs of actions conditioned on the belief profile space. This function should reflect not only the immediate value of joint action but also the expected future value. The natural way to do this is to find the value in the final timestep and back it up through time. This approach cannot be applied to POSG, because it means solving a Bayesian game in the final time step which requires as much work as solving original POSG since we need to build a Bayesian game for each previous timestep to be able to build and solve Bayesian game for the final one. Therefore the value function needs to be heuristic, which results in locally optimal policies with respect to the used heuristics. There are two biggest possible problems of this approach, the belief space Θ can be extremely large and finding a good heuristic for value function can be difficult (in some domains Q_{MDP} function had good results).

This approach was improved by MacDermed and Isbell [10]. First of all, they brought a novel integer linear program which can efficiently solve Bayesian games. Second, they present an optimal belief compression method, that can transform any DecPOMDP into a Bounded Belief DecPOMDP (BB-DecPOMDP, a special case of DecPOMDP where each player has upper bound on a number of beliefs at each timestep). This method involves agents choosing the best way to compress their own beliefs and it is divided into two phases. The first phase behaves like the original DecPOMDP. In the second one an agent's actions are decisions about how to compress its belief. Each agent must choose its next belief from a fixed number of beliefs to choose from (the number of beliefs is a free parameter). Finally, they extended results of the work of Dibangoye et.al [2] (each finite horizon DecPOMDP can be converted into a finite horizon POMDP where a probability distribution over histories is a sufficient statistics that can be used as POMDP state) to infinite horizon DecPOMDPs with bounded belief (BB-DecPOMDPs). Since DecPOMDPs can be approximated by BB-DecPOMDPs and than transformed into POMDPs, in which selecting an optimal action is equal to solve a Bayesian game, it is possible to use existing POMDP algorithms. The only difference is that instead of maximizing over actions (which is exponential for POMDPs created from BB-DecPOMDPs) a Bayesian game equivalent to the stage decision problem must be solved.

Another algorithm to solve decPOMDPs (common-payoff POSGs), which does not use transformation into a sequence of Bayesian games, was presented by Wu et.al and it is called Decentralized Rollout Sampling Policy Iteration (DecRSPI) [22]. This algorithm heuristically generates stochastic policies using an approximate policy improvement operator trained with Monte-Carlo simulation. The approximate operator simulates and evaluates a joint policy at game state by drawing some sample trajectories of the joint policy starting at particular game state. After that policy is improved by solving series of linear programs.

There were also tries to solve DecPOMDPs using dynamic programming. The first algorithm using dynamic programming was developed by Hansen et.al [4] to solve finite horizon POSGs. This algorithm is a synthesis of dynamic programming for POMDPs and iterative elimination of very weakly dominated strategies in normal-form games. It's based on the fact that every finite horizon POSG can have normal form representation. The problem of this representation is that it can be much larger than original POSG. Therefore the algorithm performs iterative elimination of dominated strategies without constructing the normal form representation. The key ingredient of the algorithm is a multi-agent dynamic backup operator which generalizes dynamic programming operator for POMDPs. First, the operator is given a set of depth-t policy trees Q_i^t and corresponding value vectors V_i^t for each agent i. The operator performs an exhaustive backup on each of the sets of trees to form Q_i^{t+1} for each agent i. Then it recursively computes value vectors V_i^{t+1}. In the second phase multi-agent operator prunes dominated strategies. Removal of policy tree results in a reduction of other agents' belief state spaces dimensionality and can be repeated until no more policy trees can be pruned from any agent's set. Since the algorithm removes very weakly dominated policies it does not guarantee optimal results for general-sum POSGs but it yields optimal strategy profiles for DecPOMDPs.

Unfortunately, this approach can solve only games with very limited horizon due to time and space complexity. These problems were addressed by the improved algorithm developed by Seuken and Zilberstein [17]. They observed that the original algorithm keeps too many

policies in memory, even though only very few of them are necessary for optimal or near-optimal behaviour, which is mostly due to its bottom-up way. The improved algorithm combines top-down heuristics (in practice MDP heuristic and random policy heuristic have proven useful) for selecting a set of most promising policy trees, bounded by the predefined maximal number of policy trees, with the original bottom-up pruning algorithm, which can be done in two different ways. First, after a full backup is performed dominated policy trees can be eliminated using the pruning algorithm. Which means no dominated policy tree will be selected by the heuristics. Second, the pruning algorithm is applied after the heuristics selected policy trees for next iteration, which leads to more memory-efficient next iteration. However, this approximate algorithm guarantees linear space and time complexity with respect to the game horizon it still grows exponentially with the size of the observation set. Therefore Seuken and Zilberstein created the improved version of their algorithm [18]. In order to reduce the number of observations, they use a technique similar to the one for selecting promising policy trees. In the first step algorithm identifies a set of belief states using top-down heuristics. Then for each belief state at timestep t the best joint policy is added. In the second step the set of most likely observations for every agent is found, bounded by the predefined maximal number of observations. For each joint observation is calculated its probability and then k most likely observations are selected based on ranking according to the probability of joint observation, where k is the maximum number of observations. For policy trees selected in the first step partial backup using only k observations for each agent is performed, which leads to a feasible number of policy trees (with missing branches).

Another dynamic programming approach to improve the original algorithm was developed by Szer and Charpillet [20]. Unlike the original one, this algorithm is top-down and works as follows. First, it is needed to generate joint policies, but generating an exhaustive set of prior policies is impractical. Therefore the strategy used for single-agent scenario (Pineau et.al [16]) is adopted. Given sets of (t-1)-step policies $Q^(t-1)_i$ for each agent i, sample from the set of possible prior policies. Those policies that are most likely to be spread out far away from each other are retained (Manhattan distance can be used as a metric). Then it is needed to generate belief states. Generating all possible beliefs states is computationally demanding due to the exponentially many possible assignments of horizon-t policies candidates. To avoid testing all possible policy subtrees they are assigned at random. Thus this algorithm computes policies based on a subset of the reachable belief states.

Besides the common-payoff POSGs there exists another subset of POSGs with attempts to solve it. This subset is called Markov Games of Incomplete Information (MGIIs) and was introduced by MacDermed et.al [12]. MGIIs generalize stochastic games and POMDPs but not decentralized models or POSGs since they can't efficiently represent private beliefs and thus MGIIs are a weaker model. An MGII is defined as a sequence of Bayesian games where both the next game and the beliefs of players in that game depends on the previous game and belief. MGIIs also have a special Markov property on players' beliefs that ensures players need to reason only about zero-level beliefs and is quite similar to the Markov property for states. Since MGIIs are by default defined as a sequence of Bayesian games they are closer to fully observable stochastic games than to POSGs. Therefore an MGII can be converted into a fully observable stochastic game and solved.

Chapter 3

Computing Correlated Equilibria in Partially Observable Stochastic Games

In this chapter, we describe our approximate algorithm to compute correlated equilibria in partially observable stochastic games. For the sake of simplicity, we will describe how the algorithm works on our running example described in Section 3.1. The rest of the chapter is structured as follows. First, we outline the high-level principle of our algorithm. Then in following sections we will describe important parts in more details. And finally in Section 3.7 we provide brief information about actual implementation.

3.1 Running Example

As we already mention we are using a pursuit-evasion game as our running example and this game represents a real-life problem when highway patrols are trying to catch a bank robber on the run.

We represent the environment of this game as a planar graph where nodes stand for the places where patrols can spot and catch the robber and edges stand for the roads connecting these places. Individual patrols are our players and the robber is represented as nature player, who uniformly randomly chooses the next place he will move to. An action is a selection of neighbouring place a player will move to.

3.2 High Level Idea

In partially observable environment players do not know the true state of the game. Instead of it they only have a distribution of possible states called belief and each belief can be represented as a state called beliefstate. When solving partially observable stochastic games the beliefstate space can be infinite. However many of these beliefstates may not be necessary for finding a solution. The main reasons can be that these unnecessary states have too low probability and have almost no impact on the solution or the outcome from these states is

so bad that we never reach them. Recognizing beliefstates needed for finding a solution is a nontrivial problem since beliefstates which seems to be promising at first can show as not as good later and vice versa. Our algorithm overcomes this hardness by an iterative extending subset of beliefset space until the values converge and approximate solution is found. The algorithm works as follows.

First, we generate a subset of the desired size from beliefstate space. We find correlated equilibrium solution using value iteration for this subset. Then we add the desired number of new beliefstates to our subset and again find correlated equilibrium solution for the extended subset. The procedure repeats until the difference between values from last two iterations is smaller than a threshold.

3.3 Generating Beliefstate Subset

When generating beliefstates from beliefstate space there are two situations which can happen.

First, we are trying to expand beliefstate we do not have computed correlated equilibrium for. In such situation just expand this state using all possible joint actions that can be played in it.

Second, we already calculated correlated equilibrium for beliefstate to expand in one of the previous iterations. In such case we expand the state with probability p according to the computed equilibrium and with probability $1-p$ using all possible joint actions that can be played. This behaviour results in expanding the parts of beliefstate space which seems promising according to actual strategy and at the same time not completely throw away the parts which are not as promising.

Setting the threshold for termination criterion can be difficult. Therefore we expand the current subset by adding newly generated states only if the difference between root state values from last two iterations is greater than 1% of the older one. It is also possible to set the minimum number of beliefstates we want to generate before terminating.

3.4 Computing Correlated Equilibrium

To show how we compute correlated equilibrium lets assume following situation. Since same constraints for selecting actions applies to all players we can use for the demonstration just point of view of one player. We are playing game showed in Figure 3.1 with two pursuers and one evader. Player 1 is at node 2 and knows that evader can be at node 3 or node 5 both with equal probability 0.5. To select the best action to perform, a player needs to know the position of player 2. Unfortunately, due to partial observability, he does not have this information and he has just belief that player 2 is either at node 0 or node 3, both with equal probability 0.5. Thus from point of view of player 1, the game can be in two possible situations and we are looking for an action for player 1 that it is the best choice for both situations.

In Figure 3.2 we can see two matrix games representing the possible state of the game from point of view of player 1 (row player). The matrix on the left corresponds to the

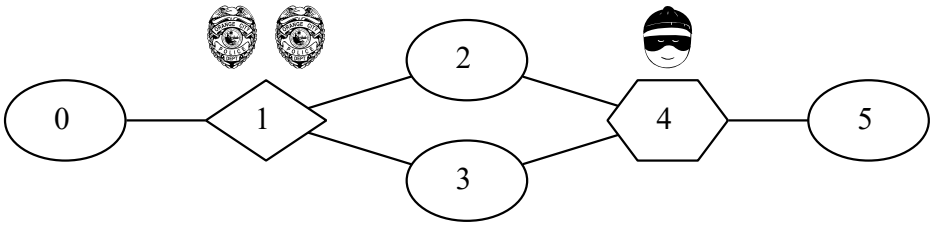


Figure 3.1: Game example

situation with player 1 in node 2 and player 2 in node 3. The right one represents the situation with player 1 in node 2 and player 2 in node 0. And as we already stated both situations are equally probable. Now we need to select an action for player 1 such that it would be the best one among all possible situations.

Since the correlation device knows the current state of the game we do not need to consider beliefs of other players. For now lets assume that the real state of the game is situation represented by the leftmost matrix. Therefore we are looking for distribution x over strategy profiles (1, 1), (1, 4), (4, 1) and (4, 4).

	1	4
1	0	5
4	10	10

	1
1	2.5
4	7.5

Figure 3.2: Belief of player 1

If this would be perfect information scenario we will just use solution concept presented in Section 2.3.1 and find a correlated equilibrium for leftmost matrix game. However, in the partially observable environment, we must consider also other states which are possible from player's point of view. This means we need to modify the original condition.

The probability of player p playing an action i at state s is $x_i^p(s) = \sum_{a \in A_{-p}(s)} x_{ia}(s)$. Lets denote the real state as s^* and the beliefstate set of player p as B^p. The set $B^p_{-s} = B^p \backslash \{s\}$ and belief of player p of being at state s is $P^p(s)$. Now lets define function $v_i^p(s)$ for computing expected value for playing action i by player p in state s:

$$v_i^p(s) = \sum_{a \in A(s)} [u_{ia}^p + \gamma c_{ia}^p(s)] x_{ia} \tag{3.1}$$

When evaluating function $v_i^p(s)$ for states form set $B^p_{-s^*}$ the values of x_{ia} are constants based on solution computed in previous iterations or uniform strategy. Thus expected value

for playing action i in state $s \in B^p_{-s^*}$ can be pre-computed, since they are constants too.

The modified version of constraints for each $s^* \in S$ will be:

$$P^p(s^*) \sum_{a \in A_{-p}(s^*)} [[u^p_{ia} + \gamma c^p_{ia}(s^*)] - [u^p_{ja} + \gamma c^p_{ja}(s^*)]] x_{ia} + x^p_i(s^*) \sum_{s \in B^p_{-s^*}} P^p(s)[v^p_i(s) - v^p_j(s)] \geq 0$$
$$\forall i, j \in A^p(s^*) \quad (3.2)$$

$$x^p_i(s^*) = \sum_{a \in A_{-p}(s^*)} x_{ia}(s^*) \quad (3.3)$$

$$\sum_{a \in A(s^*)} x_a = 1 \quad \forall i, j \quad (3.4)$$

$$x_a \geq 0 \qquad \forall a \in A(s^*) \quad (3.5)$$

These equations must hold for each player p from the set of players N.

If we recall our example from Figure 3.2 the conditions for player 1 will be:

$$0.5 * [[0 - 10]x_{(1,1)} + [5 - 10]x_{(1,4)}] + 0.5 * 2.5 \geq 0 \quad (3.6)$$
$$0.5 * [[10 - 0]x_{(4,1)} + [10 - 5]x_{(4,4)}] + 0.5 * 7.5 \geq 0 \quad (3.7)$$
$$x_{(1,1)} + x_{(1,4)} + x_{(4,1)} + x_{(4,4)} = 1 \quad (3.8)$$
$$x_{(1,1)} \geq 0 \quad (3.9)$$
$$x_{(1,4)} \geq 0 \quad (3.10)$$
$$x_{(4,1)} \geq 0 \quad (3.11)$$
$$x_{(4,1)} \geq 0 \quad (3.12)$$

Without setting objective function this linear program will still work. However, it will return the first correlated equilibrium found. When we do not want just any correlated equilibrium but in some manner optimal one, we can choose an objective function fulfilling our requirements. For example, when we want to find such correlated equilibrium that maximizes social welfare we must add the following objective function:

$$\max \sum_{p \in N} \sum_{a \in A(s^*)} u^p_a(s^*) x_a(s^*) \quad (3.13)$$

The objective function from Equation 3.13 guarantees to find a correlated equilibrium which maximizes social welfare. However, the solution can be unfair (one player is preferred). If we want our solution to maximize social welfare and be fair at the same time we must add different objective function. Lets substitute expected value of player p with variable $V(p) = \sum_{a \in A(s^*)} u^p_a(s^*) x_a(s^*)$ and introduced new constant w that represents how much we care about the fairness of our strategy. Then one of the possible solutions can be:

$$\max \sum_{p \in N} V(p) - w[\max_{p \in N} V(p) - \min_{p \in N} V(p)] \quad (3.14)$$

3.5 Value Iteration

In our algorithm, we use the multi-agent version of value iteration described in Section 2.2.1 with cycle detection on root state. We terminate value iteration in two situations. First when no value changed by more than 1%. Second when we detect cycle equilibria on root state.

3.6 Speeding Things up

To handle typical problems when solving partially observable stochastic games we use commonly used solutions.

3.6.1 Curse of Exponentiality

The biggest typical problem is the problem of nested belief. To find really optimal solution players should have beliefs not only about the game states but also the beliefs of other players. Unfortunately, nesting of such beliefs can be infinite. Therefore we use common solution and we assume just level 1 beliefs. Which means a player takes into account only his own belief and not the beliefs of other players.

3.6.2 States with low Probability

Another problem can be in many states with very low probability. The common solution to this is setting up a threshold and states with probability lower than the threshold are thrown away. We use this method too with the value of threshold set to 0.01.

3.6.3 Small Differences in Probabilities

We also encountered situations when we have many belief states with only small differences in probabilities of being in states. Most of the time the differences were so small that they can be spotted only by computer and in the real world they, would have no impact on the solution. To reduce those unnecessary belief states we used probabilities with the precision of two decimal places.

3.7 Implementation

We implemented our algorithm using JAVA programming language and for solving linear programs we used IBM ILOG CPLEX. In this section we provide the format of input files used for loading games and describe how and where the parameters of the program can be changed.

3.7.1 Input

Each game is loaded as an undirected planar graph from the input file. For input files we use the following format. The first line contains two numbers n and e where n is the number of nodes and e is the number of edges in the graph. Then follows e lines each of them representing an edge. These lines consist of two numbers a and b which are nodes connected by the edge. After these lines there is one line with the number of pursuers in team followed by a line with two numbers s_p and s_e where s_p is number of the node where pursuers start and s_e is the starting node of evader. The input file can be passed to the algorithm as input argument or specified as a parameter inside the code.

3.7.2 Setting Parameters

The core of the algorithm is main class Runner.java. In this class the most of parameters can be specified namely the following ones:

- **INPUT** - path to the input file (can be empty when passing input file as argument)
- **SEED** - value of random seed
- **DIFF_THRESHOLD** - minimum percentage change of value between iterations needed to proceed to next iteration
- **NEED_MIN_NUMBER_OF_STATES** - specifies whether we want to reach minimum number of generated states before termination
- **MIN_NUMBER_OF_STATES** - minimum number of states we want to generate before termination
- **THRESHOLD** - initial number of states generated
- **INCREASE** - number of states we want to add after each iteration
- **MAX_SOCIAL_WELFARE** - specifies whether we want to maximize social welfare
- **FAIRNESS** - specifies whether we want our solution to be fair
- **PRINT_STRATEGIES** - specifies whether computed strategies are printed to standard output

There are three more parameters which can be specified in other classes. In class GameState.java it is parameter **FOLLOW_RECOMMENDATION_PROBABILITY** which specifies the probability of following currently computed suggested action when expanding states. In class Info.java we can specify **THRESHOLD** parameter which is the minimum percentage change in value between iterations of value iteration needed to proceed to next iteration. And in class Solver.java we have argument **FAIR_WEIGHT**, this argument represents how much we care about the fairness of solution when maximizing social welfare and trying to find a fair solution at the same time.

Chapter 4

Experiments

In this chapter, we present the results of the experimental evaluation of the algorithm introduced in Chapter 3. First, we specify used hardware settings. Then we provide settings used for experiments. Finally, we show and discuss obtained results.

4.1 Hardware Settings

All experiments were conducted on a desktop PC running Debian GNU/Linux 9 (Stretch) 64-bit with 24 GB of RAM and eight-core processor AMD FX-8350 on its base clock 4.2 GHz. For solving linear programs we used IBM ILOG CPLEX 12.4.

4.2 Experiment Settings

All experiments were conducted on a games from our running example. We start with simple games to show that algorithm finds correct solutions proceeding to harder ones to show its scalability. In each game we use following settings. When an evader is caught a pursuer can obtain reward of 10 when he is the catcher or reward of 5 when the evader is caught by one of his teammates, cost for performing an action is -1. For computation of correlated equilibria we used discount factor 0.9. All experiments were repeated 20-times with different random seed. We used seeds: 7, 9, 11, 13, 18, 21, 23, 34, 40, 45, 57, 64, 382, 548, 666, 1492, 1815, 1997, 2033 and 1568123.

When adding newly generated beliefstates from beliefstate space we expand states according to currently computed recommended strategy with probability 0.6. We also tried other values for the probability of expanding state according to recommended strategy but we achieve the best results with this value. Setting the probability lower resulted in generating too much beliefstates outside optimal section of beliefstate space and caused not that good solutions. On the other hand, setting the probability higher than 0.6 pushes solver too hardly to a specific part of beliefstate space which could result in missing better solutions.

To be able to handle large numbers of beliefstates a little bit faster all experiments were run with VM option -XX:+UseG1GC and assigned maximum possible memory.

In the first experiment, we were looking for a solution that maximizes social welfare and is fair at the same time with parameter **FAIR_WEIGHT** set to 0.1. However, this constraint on solution drastically increases the time needed for one iteration of value iteration. Therefore in the rest of experiments, we were looking for any solution that maximizes social welfare. This approach still finds a solution that guarantees maximal social welfare but not the fairness thus expected values of players may differ.

4.3 Results

In this section, we present the results of the experimental evaluation. For each experiment, we attach the graphical representation of the used game. In each graphical representation the node where pursuers start is diamond shaped and the start of an evader is hexagon shaped. In first experiment (Subsection 4.3.1) we show how the algorithm works on real problems, what can be affected by changing some parameters and that it is capable of finding the optimal solution or at least a relatively close one. Then we will show how the algorithm behaves on harder games.

4.3.1 Experiment 1

For the first experiment we used the game in Figure 4.1 with two pursuers and one evader. This game is really simple and the optimal solution can be easily seen. Since evader can move only to nodes 2, 3 and 5 first actions of players should result in one player being in node 2 and the second one being in node 3. If they catch the evader they receive their rewards, game restarts and they perform the same action. If they do not catch the evader they know he is hiding in node 5 and in the next step he can move only to node 4. Therefore the best action is for both pursuers to move to node 4.

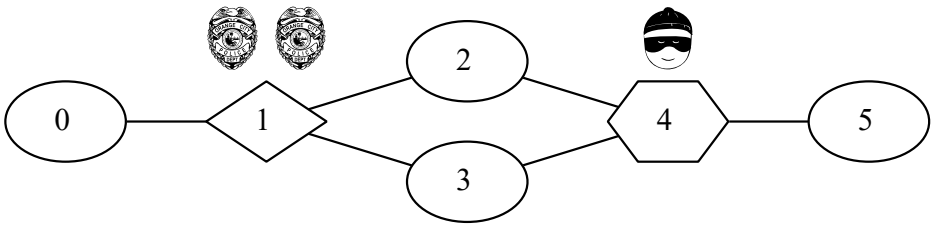


Figure 4.1: Graphical representation of experiment 1

This solution is exactly the one our algorithms finds. Since the probability of evader being in nodes 2 and 3 after one round is equal it does not matter which pursuer goes to node 2 and which one goes to node 3. Furthermore there is no need for mixed strategy either. Thus our algorithm suggests player 1 to move to node 2 and player 2 to node 3 with

probability 1 and in the case they did not catch the evader they are both suggested to move to node 4 again with probability 1.

The beliefstate space of this game is really small containing 430 beliefstates and even with generating all of them it can be solved in approximately 38.6 seconds with converged expected value 38.55 for both players.

However, it is quite obvious that we do not need all these 430 beliefstates. If we set the initial threshold for generated beliefstates to 100 and in each next iteration we add 10 newly generated beliefstates we can solve this problem in two iterations with total runtime approximately 10.15 seconds. In this scenario, we use only 110 beliefstates obtaining exactly the same strategy and the expected reward of 38.44 for both players. In this case results were the same for all random seeds we used.

And we can lower threshold and its increase furthermore. Setting the initial threshold to 40 and adding 10 newly generated states in each iteration. The number of iterations we need varies from 2 to 5 depending on the used seed. In this setting we obtained for some seeds suggested strategies which were different from the optimal one. However, the average expected value was 37.82 for both players with the minimal expected value 35.74 and the maximal expected value 38.60, which is still quite close to the optimal solution. The average runtime was 6.9 second, the fastest time was 2.69 second and the slowest one was 10.87 seconds.

Further lowering the initial value of threshold or the between iterations increase still resulted in strategies which ensure catching an evader in maximally two steps. However, the number of non-optimal suggested strategies increased. We also notice that expected values decreased but this is the result of generating a small number of beliefstates and not that these strategies are actually that bad. For example with threshold 10 and its increase 5, we need only two iterations with an average runtime of 0.73 second to find a solution. For this setting, we were able to find optimal strategy only with 9 from 20 seeds. The average expected value was 30.66 for both players, the minimal value was 23.78 and the maximal one was 35.74.

4.3.2 Experiment 2

In this experiment we used the quite similar game as in the previous one. We just extended the distance to place players need to correlate and change the starting positions. The graphical representation of game can be seen in Figure 4.2. Again we used two pursuers.

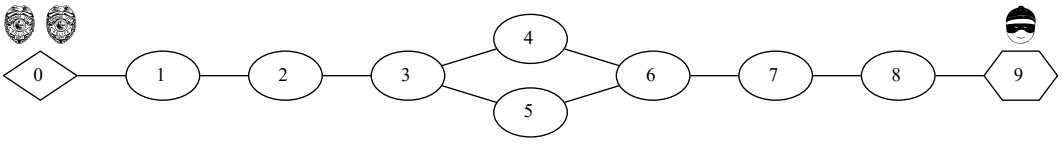


Figure 4.2: Graphical representation of experiment 2

This game still has a quite obvious solution. Pursuers need to move to node 3 as fast as possible, then one of them must go to node 4 and the second one to node 5. If the evader is not caught they move to node 6 and try to catch the evader in node 7, 8 or 9 or if they missed him on. Although this game is still quite simple the size of beliefstate space increased significantly and after 14 consecutive steps without restart there are $1,112,581$ beliefstates. When computing solution for such huge amount of states we need something about 5 hours for one iteration of value iteration. Thus it is impossible to find a solution in reasonable time.

First of all, we observe how the algorithm will behave when we use only fixed number of states and we will not add any other states. We focused on how the time needed for one iteration of value iteration will change with increasing number of states and what will be the expected values achieved by players. Since bigger amounts of beliefstates need very long time to be solved we used smaller instances. However obtained results are still very interesting and illustrate capabilities of our algorithm.

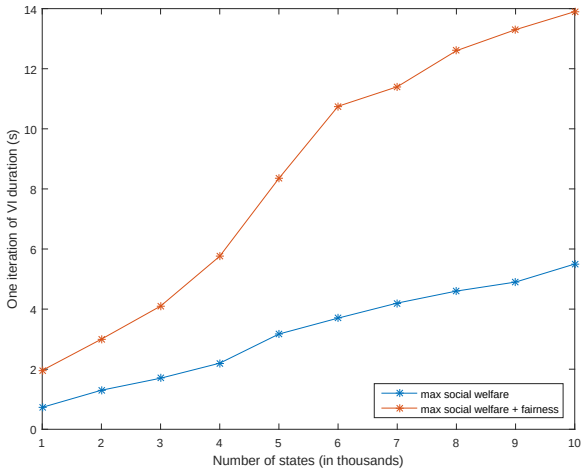
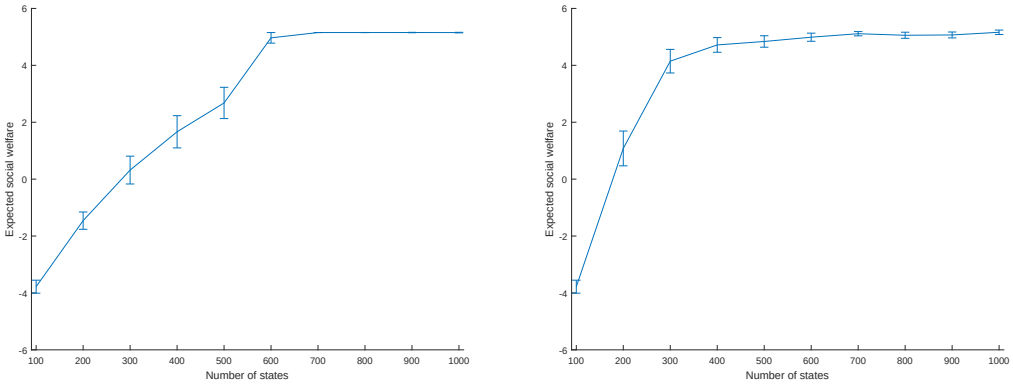


Figure 4.3: Experiment 2 - Dependency of time needed for one iteration of value iteration on number of states

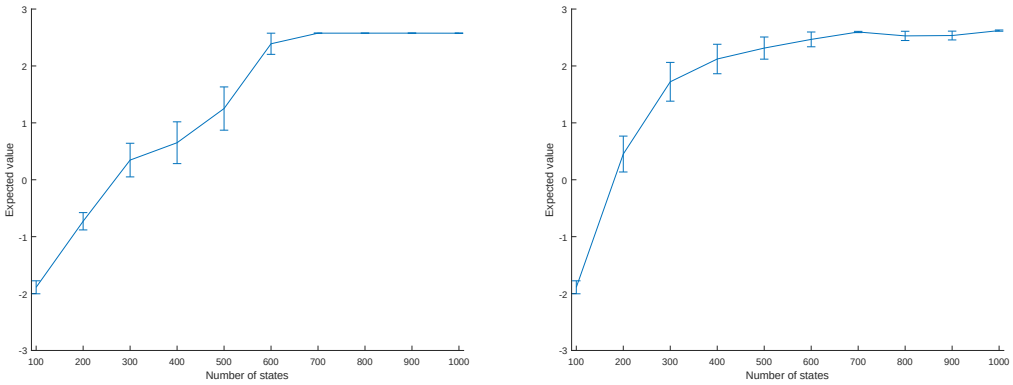
As we can see in Figure 4.3 the time required for computing one iteration of value iteration increases almost linearly with the number of states for both versions of social welfare maximization. However we can see that insisting on the fairness of solution results huge increase in time needed, thus we decided to find solutions without fairness. This can result in a difference between expected values for individual players but the social welfare remains the same as in case of fair solutions. We experienced the same behaviour for all games we have tested during our experiments.

On Figure 4.4, Figure 4.5 and Figure 4.5 we can see that for iterative adding newly generated states the expected values for both players and social welfare converge faster then for fixed sized subsets of beliefstates.



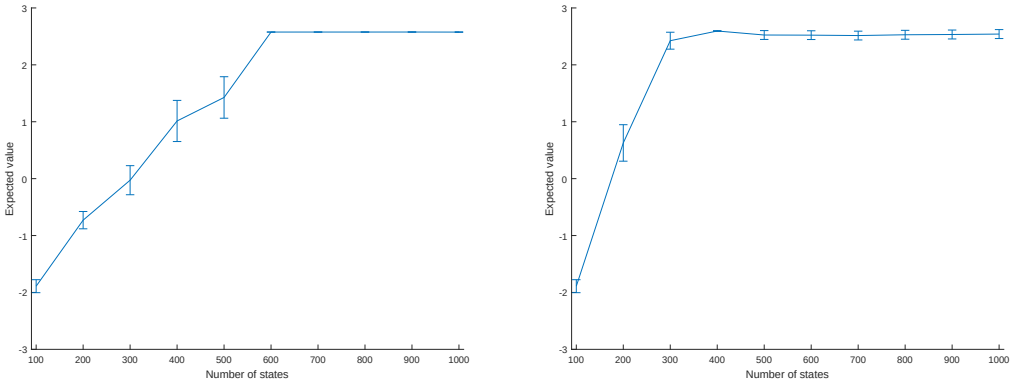
(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.4: Experiment 2 - Convergence of expected social welfare



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.5: Experiment 2 - Convergence of expected value for player 1



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.6: Experiment 2 - Convergence of expected value for player 2

4.3.3 Experiment 3

For our third experiment we used game showed in Figure 4.7 with two pursuers and one evader. Again the best strategy to follow can be easily spotted but it is not completely intuitive. Since the evader can escape from node 4 to part of the graph where catching him becomes very difficult pursuers need to make sure that he cannot do this. In perfect information setting it would be sufficient for only one player to move to node 4 and the second one could move either to node 1 or node 3. However, in a partially observable setting when a player does not move to node 4 the possibility of evader escaping remains. Therefore first both players should move to node 4 to make themselves sure that evader cannot escape. This means that in the situation they did not catch the evader in node 4 the evader must be either in node 1 or node 3 both with equal probability 0.5. Thus after next move the evader will be in node 0 with probability 0.5 or in the node 2 with the same probability. Because of this the second suggestion is moving to node 0 for one player and moving to node 2 for the second one.

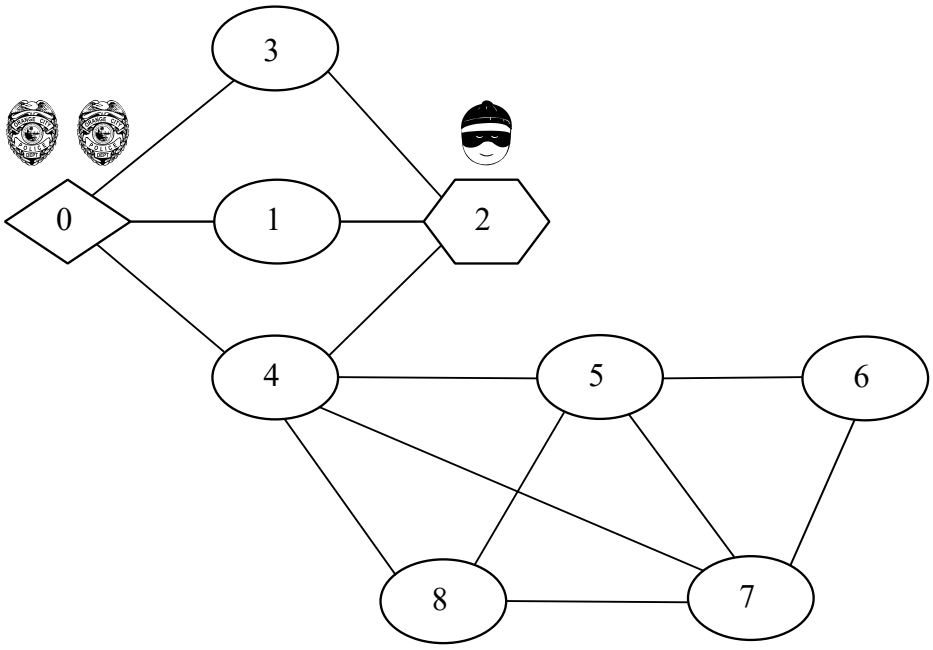
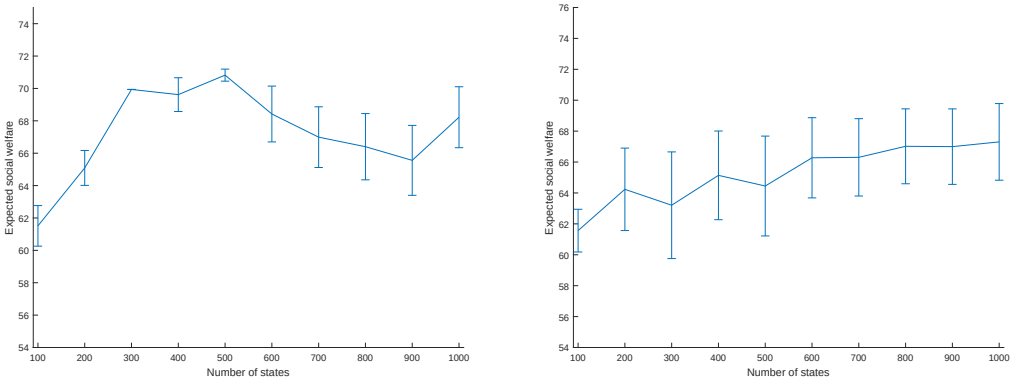


Figure 4.7: Graphical representation of experiment 3

The same solution was found using our algorithm. Again as in the case of previous experiments we present convergence graphs for expected values and expected social welfare with a fixed number of bleliefstates and iterative adding beliefstates between iterations. Since pursuers need at maximum two moves to catch the evader the optimal strategy we do not need many beliefstates because adding additional states does not bring much more.

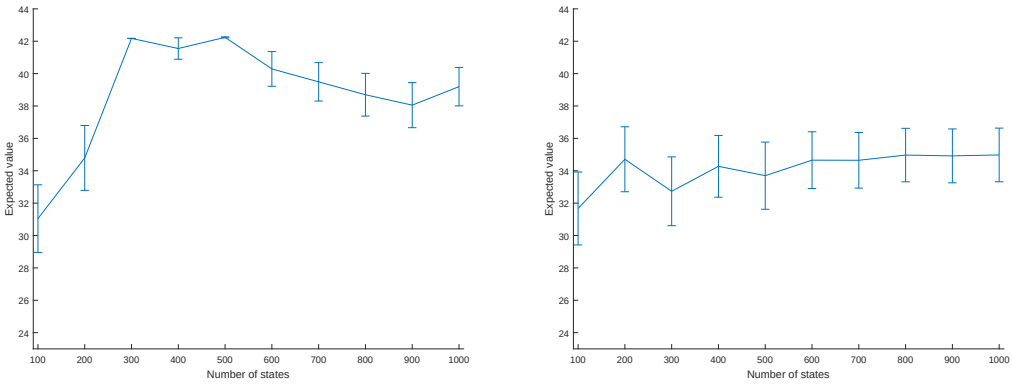
First lets focus on the convergence of expected social welfare. Comparing results obtained for a fixed number of beliefstates (Figure 4.8a) and iteratively adding newly generated states (Figure 4.8b) we can see that for both the expected value somewhat converges. However, the properties of convergence for iterative adding are better. As we can see the convergence

is much smoother, faster and we can actually see the values converging. In case of a fixed number of beliefstates the convergence is not that smooth. However, the obtained results are pretty close to results obtained using iterative adding. The reason for such good results for completely random generating a fixed number of states is that this particular game does not need many states for finding the optimal strategy.



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

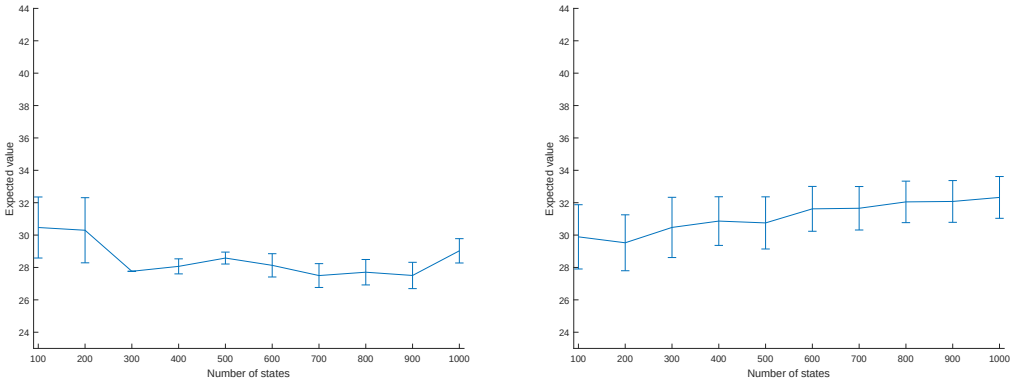
Figure 4.8: Experiment 3 - Convergence of expected social welfare



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.9: Experiment 3 - Convergence of expected value for player 1

As we can see in Figure 4.9 and Figure 4.10 the expected values converges in a similar way like expected social welfare. However, we can spot bigger standard error for iterative adding newly generated states. After deeper analysis of obtained results we found out that the cause to this is that the preferred player differs with seed.



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.10: Experiment 3 - Convergence of expected value for player 2

4.3.4 Experiment 4

In this experiment we want to show that our algorithm is capable of finding some solution even for more complex games than we already presented. To achieve this we used the game in Figure 4.11. As in previous experiments we use setting with two pursuers and one evader.

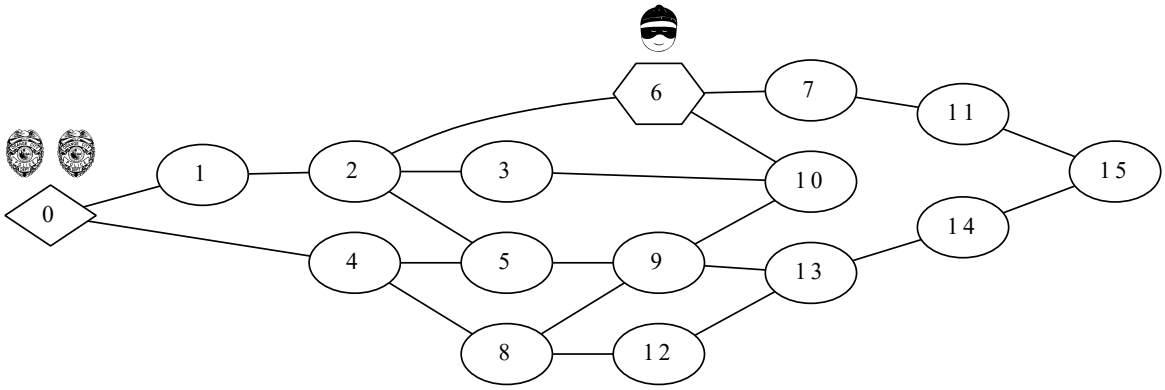
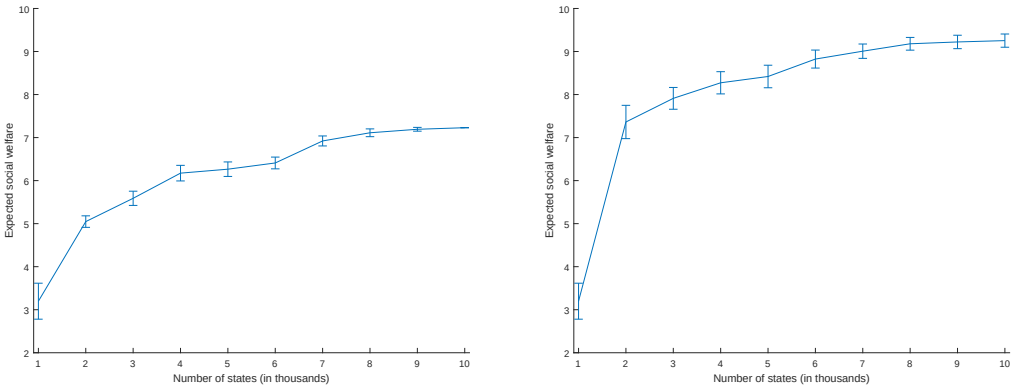


Figure 4.11: Graphical representation of experiment 4

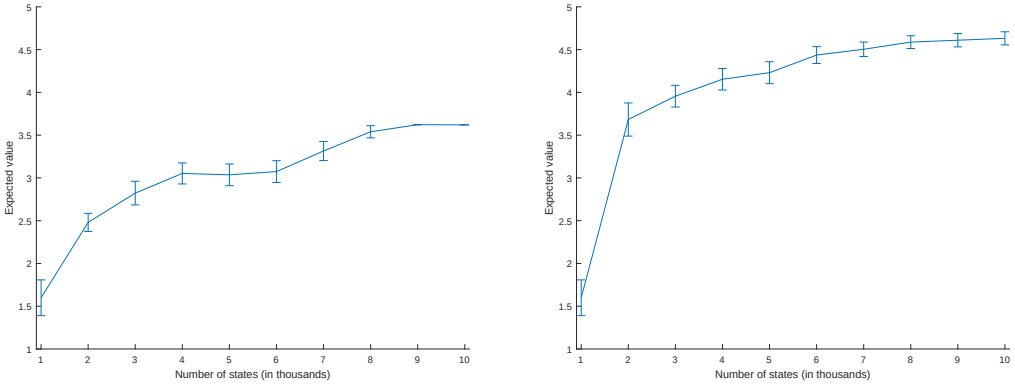
This game is too complex for a human to find the solution. Thus we cannot evaluate the quality of strategy better way then looking at expected values. As in previous experiments we show the convergence graphs of expected social welfare and values for the fixed number of beliefstates and iteratively increasing number of beliefstates.

As we can see for this game iterative adding of newly generated states behave significantly better than using fixed sized subsets of beliefstates. The expected social welfare and expected values for both players converges faster and to better rewards when using iterative adding. The main reason for this is that the iterative adding allow us to observe a longer horizon with the same number of states compared to fixed sized subsets.



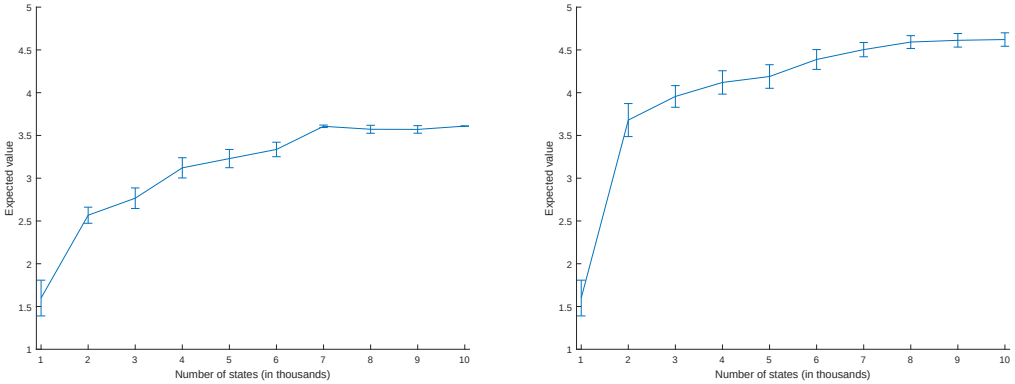
(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.12: Experiment 4 - Convergence of expected social welfare



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.13: Experiment 4 - Convergence of expected value for player 1



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.14: Experiment 4 - Convergence of expected value for player 2

4.3.5 Experiment 5

We already showed that our algorithm works in scenarios with two pursuers. Now we would like to show that it works for higher numbers of pursuers too. In this experiment we use the game presented in Figure 4.15. This game is exactly the same as the one we used in the previous experiment but this time there are three pursuers instead of two.

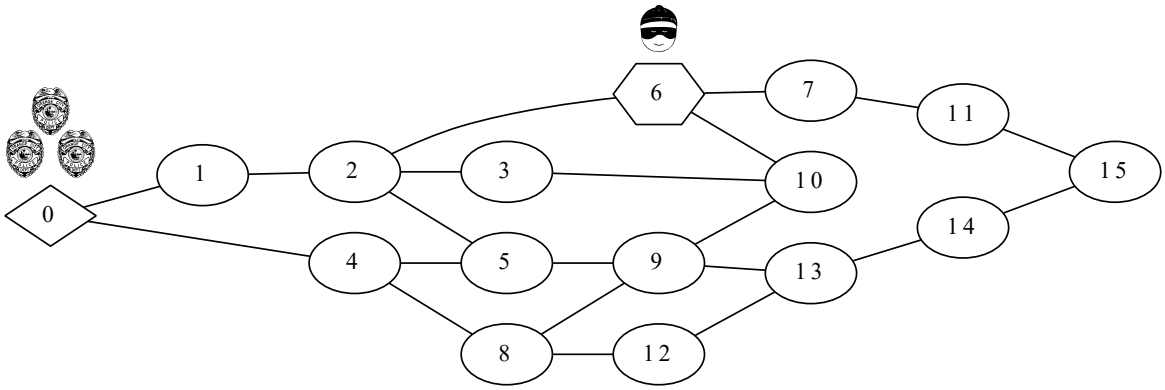
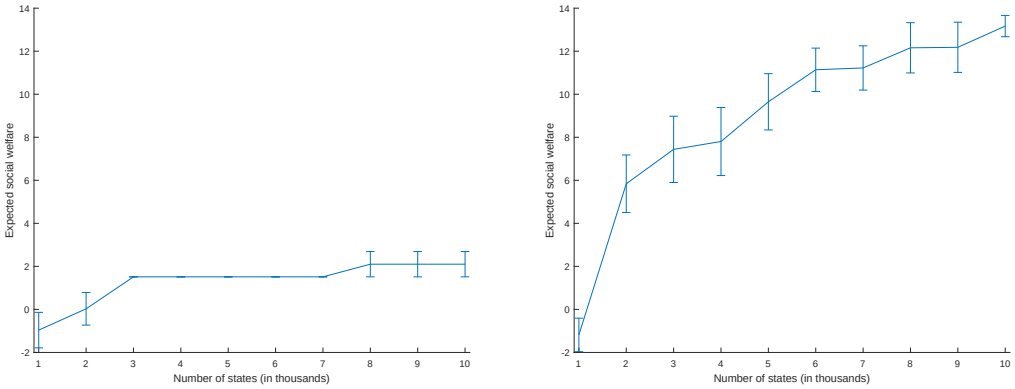


Figure 4.15: Graphical representation of experiment 5

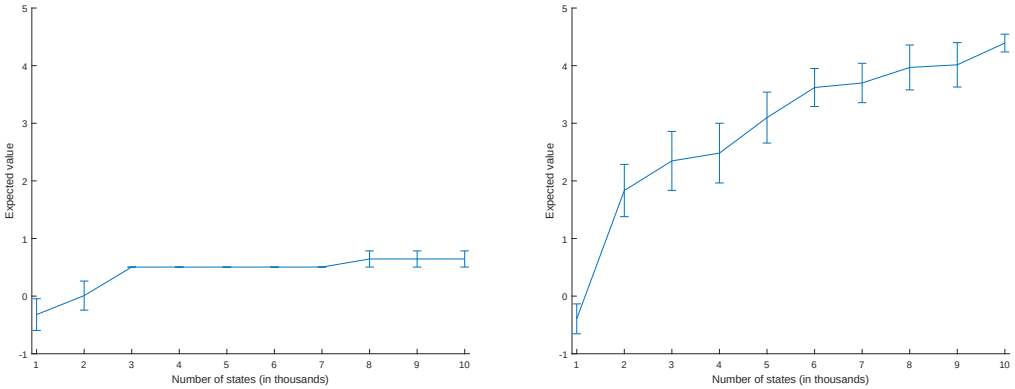
Again we provide the convergence graphs of expected social welfare and values for the fixed number of beliefstates and iteratively increasing number of beliefstates.



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

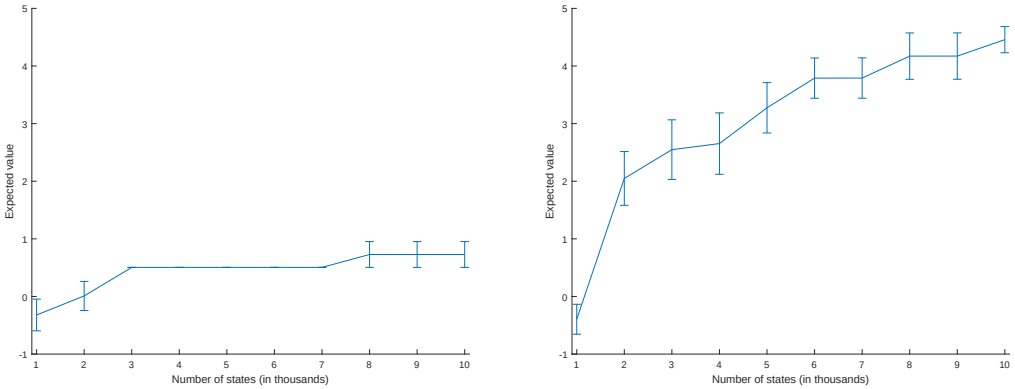
Figure 4.16: Experiment 5 - Convergence of expected social welfare

As we can seen when we increase the number of players the difference between using fixed number of beliefstates and iteratively adding newly states becomes more visible. On Figure 4.16, Figure 4.17, Figure 4.18 and Figure 4.19 we see that expected values for social welfare and player rewards converges much faster and to higher values for iterative adding. That is again because iterative adding allows us to observer longer horizon with the same number of beliefstates generated.



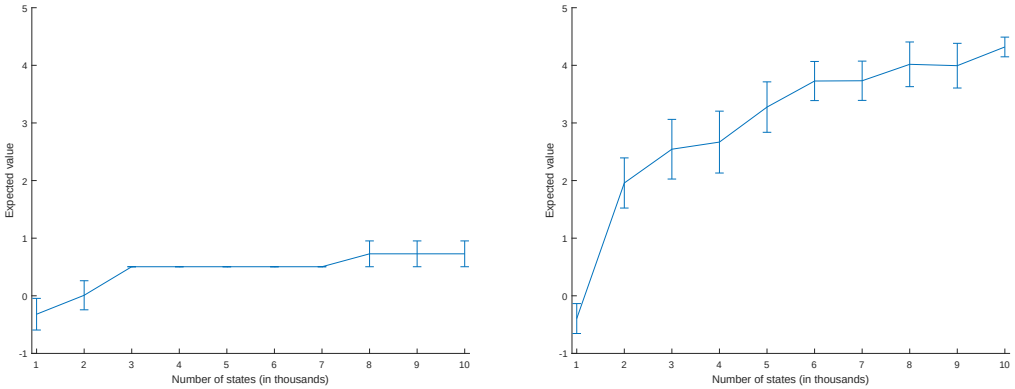
(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.17: Experiment 5 - Convergence of expected value for player 1



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.18: Experiment 5 - Convergence of expected value for player 2



(a) Fixed number of beliefstates (b) Iterative adding new beliefstates

Figure 4.19: Experiment 5 - Convergence of expected value for player 3

Chapter 5

Conclusion and Future Work

In the real world we have to deal with situations requiring the cooperation of participating agents keeping their rationality. These problems are well addressed by the game-theoretical concept of correlated equilibrium. There exist some works dealing with the problem of computing correlated equilibria in partially observable stochastic games but they assume only specific subsets of these games (mainly partially observable stochastic games with common payoffs). Thus, so far there is no algorithm capable of computing correlated equilibria in general partially observable stochastic games.

In this work we present an overview of current literature on computing correlated equilibria in stochastic games and partially observable stochastic games. We focus on computing correlated equilibria in partially observable stochastic games and provide an approximate algorithm. To the best of our knowledge, it is the first approximate algorithm for general partially observable stochastic games. Even though we have no theoretical guarantees of its convergence, in experimental evaluation we empirically show that our algorithm is capable of finding reasonable solutions for computing correlated equilibria in partially observable stochastic games and the expected values for these solutions converge. Furthermore, for problems with known optimal solution we showed that our algorithm can find this solution using only a small fraction of beliefstate space.

There are many ways for future research. Since we have no theoretical guarantees of convergence of our algorithm finding them is the first target of the future research. Another target should be to confirm the good results achieved on our running example also on other problems represented as partially observable. There can be also done some improvements to the algorithm itself. For example implementing more sophisticated for handling cyclic equilibria. We can also focus on time and memory efficiency of our algorithm and try to achieve some improvements in this area, probably by using some kind of pruning. It is also possible to test how would the algorithm behave when we introduce some heuristics, either for better estimate of initial values or for deciding which parts of beliefstate set are the most promising ones and thus worth to explore.

Bibliography

[1] BOWLING, M. Convergence problems of general-sum multiagent reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning*. 2000, s. 89–94.

[2] DIBANGOYE, J. S. et al. Optimally Solving Dec-POMDPs as Continuous-State MDPs. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI-13)*. 2013.

[3] EMERY-MONTEMERLO, R. et al. Approximate solutions for partially observable stochastic games with common payoffs. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*. 2004, 1, s. 136–143.

[4] HANSEN, E. A. – BERNSTEIN, D. S. – ZILBERSTEIN, S. Dynamic programming for partially observable stochastic games. *19th National Conference on Artificial Intelligence*. 2004, s. 709–715.

[5] HU, J. – WELLMAN, M. P. Multiagent reinforcement learning: theoretical framework and an algorithm. *Proceedings of the 15th International Conference on Machine Learning*. 1998, s. 242–250.

[6] LITTMAN, M. L. Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the 11th International Conference on Machine Learning*. 1994, s. 157–163.

[7] LITTMAN, M. L. Friend-or-foe Q-learning in general-sum games. *Proceedings of the 18th International Conference on Machine Learning*. 2001, s. 322–328.

[8] LITTMAN, M. L. – SZEPESVARI, C. A generalized reinforcement-learning model: Convergence and applications. *Proceedings of the 13th International Conference on Machine Learning*. 1996, s. 310–318.

[9] LITTMAN, M. L. – ZINKEVICH, M. – GREENWALD, A. Cyclic Equilibria in Markov Games. *Advances in Neural Information Processing Systems*. 2006, s. 1641–1648.

[10] MACDERMED, L. C. – ISBELL, C. L. Point based value iteration with optimal belief compression for Dec-POMDPs. *Advances in Neural Information Processing Systems*. 2013, s. 100–108.

[11] MACDERMED, L. C. – ISBELL, C. L. Solving stochastic games. *Advances in Neural Information Processing Systems*. 2009, s. 1186–1194.

[12] MACDERMED, L. C. – ISBELL, C. L. – WEISS, L. Markov Games of Incomplete Information for Multi-Agent Reinforcement Learning. *Interactive Decision Theory and Game Theory*. 2011.

[13] MURRAY, C. – GORDON, G. J. Multi-robot negotiation: approximating the set of subgame perfect equilibria in general-sum stochastic games. *Advances in neural information processing systems 19*. 2007, s. 1001–1008.

[14] MURRAY, C. – GORDON, G. J. Finding correlated equilibria in general sum stochastic games. *Technical Report*. 2007.

[15] PAPADIMITRIOU, C. H. – ROUGHGARDEN, T. Computing Correlated Equilibria in Multi-player Games. *Journal of the ACM*. 2008, 55, 3, s. 1–29.

[16] PINEAU, J. – GORDON, G. – THRUN, S. Point-based value iteration: An anytime algorithm for POMDPs. *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. 2003, s. 1025–1032.

[17] SEUKEN, S. – ZILBERSTEIN, S. Memory-bounded dynamic programming for DEC-POMDPs. *Proceedings of the 20th International Joint Conference on Artificial Intelligence*. 2007, s. 2009–2015.

[18] SEUKEN, S. – ZILBERSTEIN, S. Improved memory-bounded dynamic programming for decentralized POMDPs. *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*. 2007, s. 344–351.

[19] SHAPLEY, L. S. Stochastic Games. *Proc. Nat. Acad. Sci. U.S.A.* 1953, 39, s. 1095–1100.

[20] SZER, D. – CHARPILLET, F. Point-based dynamic programming for DEC-POMDPs. *Proceedings of the 21st National Conference on Artificial Intelligence*. 2006, s. 1233–1238.

[21] TOMASEK, P. Solving Stochastic Games with Imperfect Information. *Bachelor's Project*. 2016.

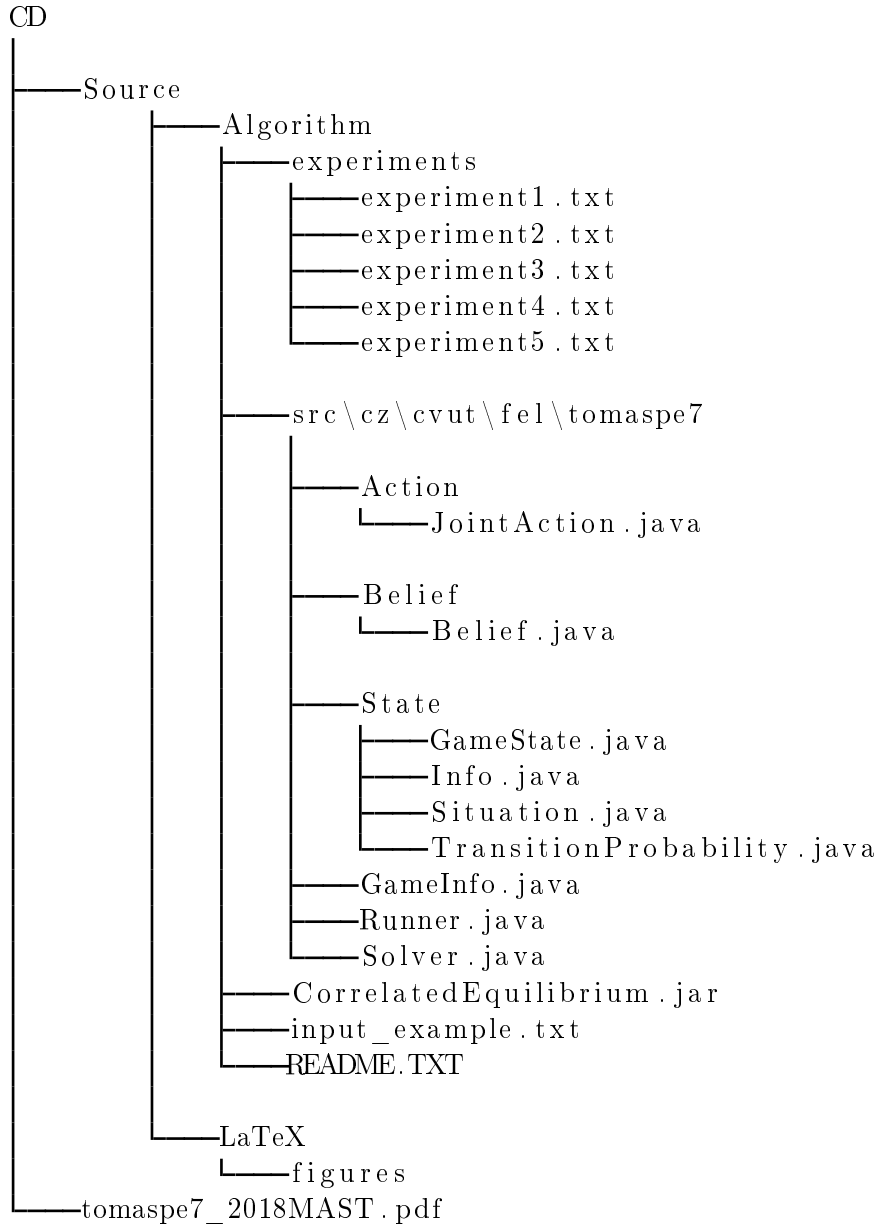
[22] WU, F. – ZILBERSTEIN, S. – CHEN, X. Rollout sampling policy iteration for decentralized POMDPs. *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*. 2010, s. 666–673.

Appendix A

CD Content

Attached CD contains pdf version of this thesis, LaTeXsource code of this thesis and the the java implementation of algorithm presented in this work. More detailed structure of CD content is presented on the next page.

- tomaspe7_2018MAST.pdf - pdf version of this thesis
- Algorithm - folder containing implementation of introduced algorithm
- experiments - folder containing all inputs used for experiments
- src - folder containing sources of our algorithm
- JointAction.java - representation of join action
- Belief.java - representation of belief of one player
- GameState.java - representation of beliefstate
- Info.java - class that holds all information needed about a game state
- Situtaion.java - representation of possible situation in game
- TransitionProbability.java - class holding transitions with their probabilities for a beliefstate
- GameInfo.java - representation of game instance
- Runner.java - implementation of our algorithm
- Solver.java - implementation of solver used for computing correlated equilibria
- CorrelatedEquilibrium.jar - runnable version of our algorithm
- input_example.txt - example of input file for CorrelatedEquilibrium.jar
- README.txt - manual for running CorrelatedEquilibrium.jar
- LaTex - folder containing LaTeX sources of this thesis
- figures - folder containing all figures used in this thesis


CD
Source
Algorithm
experiments
experiment1.txt
experiment2.txt
experiment3.txt
experiment4.txt
experiment5.txt
src\cz\cvut\fel\tomaspe7
Action
JointAction.java
Belief
Belief.java
State
GameState.java
Info.java
Situation.java
TransitionProbability.java
GameInfo.java
Runner.java
Solver.java
CorrelatedEquilibrium.jar
input_example.txt
README.TXT
LaTeX
figures
tomaspe7_2018MAST.pdf