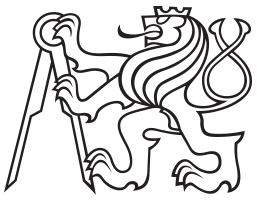
Bachelor Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Strategy Generation for Partially Observable Stochastic Games

Erik Vaknin

Supervisor: Mrg. Branislav Bošanský, Ph.D.
May 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Vaknin Erik** Personal ID number: **466195**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Strategy Generation for Partially Observable Stochastic Games

Bachelor's thesis title in Czech:

Generování strategií pro řešení částečně pozorovatelných stochastických her

Guidelines:

One-Sided Partially Observable Stochastic Games are dynamic games with infinite horizon where only one player has imperfect information and the opponent has full information. Such games can be applied in many scenarios -- for example, a known board game called Scotland Yard where a group of agents is trying to capture an evader having only limited observations about the movement of the evader. However, the scalability of recently introduced algorithm PG-HSVI [1] is insufficient for these games since there are many actions for one player. In the literature, strategy-generation techniques are a popular method for solving games with an exponentially large set of strategies. The goal of the student is to:
1. Get familiar with the algorithm PG-HSVI.
2. Design a modification of PG-HSVI based on strategy-generation techniques while preserving theoretical convergence guarantees.
3. Design an approximate/heuristic variant of the strategy-generation PG-HSVI.
4. Implement these two methods and experimentally compare the performance of the proposed modifications with the original algorithm on a simplified variant of the Scotland Yard game.

Bibliography / sources:

[1] Horák, K., Bošanský, B., & Pěchouček, M. (2017). Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In AAAI (pp. 558-564).
[2] McMahan, H. B., Gordon, G. J., & Blum, A. (2003). Planning in the presence of cost functions controlled by an adversary. In Proceedings of the 20th International Conference on Machine Learning (ICML-03) (pp. 536-543).
[3] Bošanský, B., Kiekintveld, C., Lisý, V., & Pěchouček, M. (2014). An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. Journal of Artificial Intelligence Research, 51, 829-866.

Name and workplace of bachelor's thesis supervisor:

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

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

Date of bachelor's thesis assignment: **11.01.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Mgr. Branislav Bošanský, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

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

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgements

I am grateful to Mrg. Branislav Bošanský, Ph.D., lecturer, in the Artificial Intelligence Center, FEE. I am thankful and indebted to him for sharing expertise and sincere and valuable guidance.

I must also express my gratitude to my mother and to my fiancee for providing me with support and encouragement throughout my years of study and through the process of researching and working on this thesis. Thank you.

Erik Vaknin

Declaration

I declare that the 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.

Prague, 23. May 2019

Abstract

The aim of this thesis is to design and implement a modification of algorithm PG-HSVI [1] (that was designed to solve partially observable stochastic games [1]), that might improve its ability to solve larger problems. For this we applied an incremental strategy generation method. The ISG-PG-HSVI algorithm introduced in this thesis starts with simplifying the original game by removing some of the available actions for one of the players. Then it solves this simplified variant of the game and saves the result. After that it adds some of the actions back and solves this game, while reusing the results from the previous iteration. This repeats until the original game is solved. Furthermore we designed and implemented a heuristic with the aim to show, whether it matters, which actions are added to the game beforehand. In the end we present the results of comparing these algorithms in experiments.

Keywords: game theory, partially observable stochastic games, strategy generation, heuristic search value iteration

Supervisor: Mrg. Branislav Bošanský, Ph.D.

Abstrakt

Cílem této teze je navrhnout a naimplementovat modifikaci algoritmu PG-HSVI [1] (který byl navrhnut k řešení částečně pozorovatelných stochastických her [1]), která by mohla vylepšit jeho schopnost řešit větší problémy. K tomu jsme využili metody incrementálního generování strategií. Algoritmus ISG-PG-HSVI představený v této tezi začíná zjednodušením původní hry tak, že odstraní některé z dostupných akcí pro jednoho z hráčů. Poté vyřeší tuto zjednodušenou variantu hry a výsledek uloží. Následně přidá do hry zpět některé z chybějících akcí a vyřeší tuto novou hru s využitím uložených výsledků z přechozí iterace. Toto se opakuje, dokud není vyřešena celá původní hra. Dále jsme navrhli a naimplementovali heuristiku s cílem ukázat, zda záleží na tom, které akce jsou přidány do hry přednostně. Na konci předkládáme výsledky srovnání těchto algoritmů v experimentech.

Klíčová slova: teorie her, částečně pozorovatelné stochastické hry, generování strategií, heuristic search value iteration

Překlad názvu: Generování strategií pro řešení částečně pozorovatelných stochastických her

Contents

Figures

Tables

Chapter 1

Introduction

One-sided partially observable stochastic games (POSG) are dynamic games with infinite horizon [1]. One-sided partial observability means that only one player has not perfect information, the other one has perfect information about the game. Games with infinite horizon are games, in which players can not expect the game to end after $n \in N$ steps, in other words there is at least one way to play the game so that it never ends. This game concept can be applied to a well known board game Scotland Yard. In this game there is a group of police officers (player 1) chasing a thief (player 2) while having just a little information about his transports.

These problems are studied and solved by game theory. Some people might think, that game theory is impractical, purely theoretical, but we should point out that game theory is broadly used in many practical areas. For example in cyber security, game theory is used for detection and prevention of intrusion, or to improve security of self-organised networks and cloud computing [6]. Furthermore biologist found out, that many different animal species learned benefits of cooperation, and these behaviors can be modeled by game theory. And there are more fields that utilize findings of game theory [7].

There is a recently introduced algorithm PG-HSVI [1], that can solve two-player POSGs, but his scalability is not good enough for solving a game such as Scotland Yard efficiently. The reason is that there are too many available actions for players in these games, i.e. the number of possible next states grows rapidly with each step of the game. For example in Scotland Yard with a map being a grid 4x5 and with player 1 having 3 police officers, the average amount of possible actions to play by a player 1 is about 70, and about 4 playable by player 2. That means that if we are in a certain state of the game, there are likely (depends on the state) over 3 septillion ($3 \cdot 10^{24}$) possible states to be in after ten moves.

But nowadays strategy-generation methods are often used for solving

games with exponentially large set of strategies, therefore it is interesting to investigate the possibilities for using this method for one-sided POSGs. That is the reason why we decided to design and implement a modification of PG-HSVI based on a incremental strategy generation method. That means the algorithm takes a game as a problem, solves its very simplified variant, takes the results and reuses them while solving a bit less simplified variant. This repeats until it solves the original full game. In our case the simplification means, that the algorithm gives just a small part of all originally available actions to the pursuers, and then it gradually increases the current set of actions.

Also we decided to design and implement a heuristic variant of the incremental strategy generation variant PG-HSVI. We conceived a heuristic, that, before running incremental strategy generation PG-HSVI, it runs PG-HSVI many times on very simplified variants of the game the same way as described above, but each time generates a different set of actions and then remembers the best set, which then uses as a start set of actions for incremental strategy generation PG-HSVI.

In the beginning of this thesis we go through the theory behind these algorithms, define the concepts. Then we define the Scotland Yard game as a partially observable stochastic game. The next part is about necessary implementation of generator of Scotland Yard instances. After this we present our modification of PG-HSVI and added heuristic. After all designs and implementations were done, we designed and performed experiments, in which we compared these two new approaches together with the original PG-HSVI on a feasible variant of the Scotland Yard game. In the last part of the thesis we analyze the results.

Chapter 2

Theory

In the theoretical part we will look into how the Heuristic search value iteration algorithm (HSVI) works. HSVI was designed to solve partially observable Markov decision processes (POMDP), therefore we need to define them first. Later we will define the partially observable stochastic games before exploring the variant of HSVI designed to solve them (PG-HSVI). In the last two parts of theoretical part we present incremental strategy generation modification of the PG-HSVI and added heuristic to this modification.

2.1 Partially observable Markov decision process (POMDP) [2]

In this section we define partially observable Markov decision process and then demonstrate an example of such process. Partially observable Markov decision process is a generalized Markov decision process (MDP). The system dynamics are the same as in the MDP, nevertheless the agent can not observe current state the game is in. Instead the agent must manage a probability distribution over a set of potential states. The only available information for the agent are the system dynamics, observations and their probabilities.

2.1.1 Definition [2]

POMDP is a tuple $(S, A, T, R, O, \Omega, \gamma, b_0)$, where S is a finite set of states, A is a finite set of actions, T is a stochastic transition function $(T : S \times A \times S \to [0, 1])$, R is a reward function $(R : S \times A \to \mathbb{R})$, O is a finite set of observations, Ω is a stochastic observation function, γ is a discount factor $(0 \leq \gamma < 1)$ and b_0 is an agent's initial belief. The belief is a discrete probability distribution over the states. It's value for each state expresses the

agent's belief, that the game is in the particular state.

Let s_t be state at time t, a_t action at time t, o_t observation at time t. Then

$$b_0(s) = p(s_0 = s)$$

$$T(s, a, s') = p(s_{t+1} = s' \mid s_t = s, a_t = a)$$

$$\Omega(s, a, o) = p(o_t = o \mid s_{t+1} = s, a_t = a)$$

At each time t the agent is in a state $s_t \in S$ and takes an available action $a_t \in A$. That causes getting at time $t + 1$ to a state $s_{t+1} \in S$ and receiving an observation $o \in O$ and reward based on T, Ω, R.

The goal of the agent is to maximize his expected future reward $\sum_{t=0}^{\infty} \gamma^t r_t$

At time t the agent doesn't know the state he is in. The agent knows just an initial belief b_0, history of actions he made, and observations he got during the process. Based on these information, agent can play optimally by adjusting his strategy at each step of the game based on his current belief.

2.1.2 Example - The tiger problem

In this problem, an agent is in a room with two doors and faces a decision. Agent must pick either opening left door or right door. Behind one of this door is located tiger. Agent wants to get out as soon as possible, but mainly wants to avoid opening the door with the tiger. He has also an option to listen to tiger's roaring, which takes time, but can indicate where the tiger could be. The following assignments show that this problem can be formulated as a POMDP according to definition.

$S = \{\text{tiger left} = s_l, \text{ tiger right} = s_r, \text{ end}= s_e\}$

$A = \{\text{open left} = a_{ol}, \text{ open right} = a_{or}, \text{ listen} = a_l\}$

$O = \{\text{hearing tiger on the left} = o_l, \text{ hearing tiger on the right} = o_r, \text{ end} = o_e\}$

$R = \{[s_l, a_{or}] : +10, [s_r, a_{ol}] : +10, [s_l, a_{ol}] : -100, [s_r, a_{or}] : -100, [*, a_l] : -1\}$

$T = \{[*, a_{ol}/a_{or}] : s_e, [s_l/s_r, a_l] : s_l/s_r\}$

$\Omega = \{[s_l, a_l, o_l] : 0.85, [s_l, a_l, o_r] : 0.15, [s_r, a_l, o_r] : 0.85, [s_r, a_l, o_l] : 0.15, [*, a_{ol}/a_{or}, o_e] : 1\}$

There is 85% chance to hear the tiger from the door, where he really is.

$\gamma = 1$

$b_0 = 0.5$

The agent does not know, in which state he is. He only has a belief about that. Belief b_0 says, that there is 50% chance, that the tiger is on the left, and 50% chance, that the tiger is on the right. In this case we can imagine the belief as a number from interval $[0; 1]$ (since there are n states, the set of possible beliefs is an object in a dimension $n - 1$). Let belief $b = 0$ mean 100% certainty that tiger is on the left, belief $b = 1$ mean 100% certainty that tiger is on the right.

It is obvious, that if the agent wants to play optimally, he must react on the observations he obtains after each action, in other words he must adjust his belief about in which state he is. At the start of the game his belief is $b = b_0 = 0.5$. For example he chooses to listen first and he hears the roaring from the left. Than he his belief after this is $b_1 = 85\%$. He can whenever choose to open any door. If he continues to listen, he is changing his belief every time he does so based on probability.

Imagine a new game. In each iteration of the game t the agent has some belief b_t about the game. Since we know both stochastic transition and observation functions, we can calculate the value of each possible action for any belief b_t. Let's call $v(b_t, a)$ the value of action a in belief b_t.

$$v(b_t, a_0) = 10(1 - b_t) - 100b_t = 10 - 110b_t$$

$$v(b_t, a_1) = -100(1 - b_t) + 10b_t = -100 + 110b_t$$

$$v(b_t, a_2) = -1$$

From these equations it is clear, that for beliefs $b \leq 0.1$ the best decision is to open the right door and for beliefs $b \geq 0.9$ the best decision is to open the left door. In all the other cases the best choice is to listen. These value functions of actions (1-step strategies) can be seen in figure 2.1. Now is there a generally optimal finite sequence of actions including reacting on observations, that ends the game (strategy)? Example of such strategy could be: "Listen, then if heard tiger on the left, open the right door, if heard tiger on the right, open the left door". This particular strategy is 2 step strategy. To answer the question - no, there is no such strategy. To outline the proof - for every n-step strategy we can simply find a game, for which the strategy is not optimal. In that game if the agent got an observation o_l, the next observation he receives is o_r and vice versa.

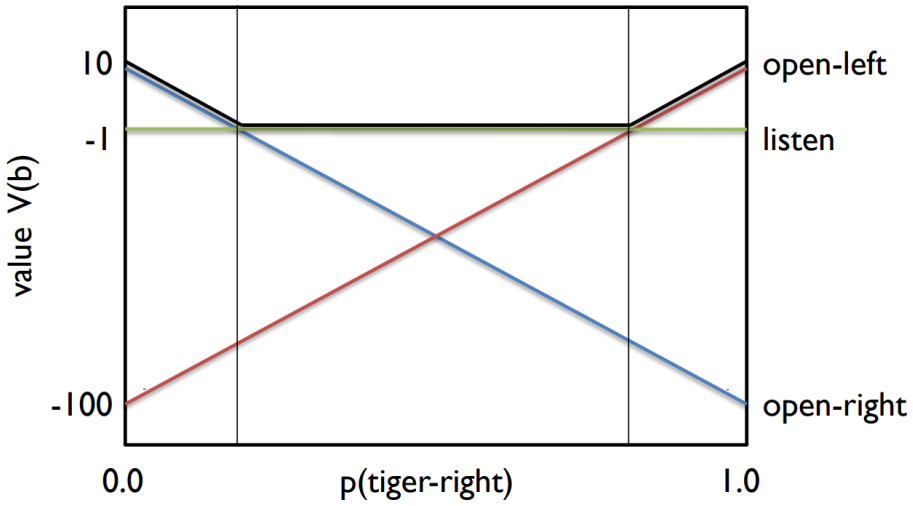


Figure 2.1: Value functions of 1-step strategies [5]

2.2 Heuristic search value iteration (HSVI) [2]

Before going through the HSVI algorithm it is necessary to define a value of a strategy and value of a game.

Definition: Value of a strategy π_1 of player 1 is a function V_{π_1}, that assigns an expected reward $V_{\pi_1}(b_0)$ of player 1 according to belief, assuming that after P1 playing according to the strategy π_1 and player 2 always playing a best response.

Lemma: Value of a strategy is always linearly dependant on belief [1].

Definition: Value of a game G is a function V^*, that assigns a maximum value $V^*(b_0)$ of all possible strategies of player 1, i.e. it is a value of the best strategy of player 1.

PODMPs are often intractable to solve exactly. That caused formation of algorithms computing approximate solutions for POMDPs [8]. HSVI is one of them. It combines representation of piece-wise linear convex value function and heuristic search procedures. HSVI keeps lower and upper bounds (LB and UB) of the value function V^*. The essential action, that HSVI does, is updating bounds of optimal value function at a certain belief. The belief selection is based on heuristics. The strategy used for exploration is depth-first search.

Let's call the LB function V^-, the UB function V^+. Then $\hat{V}$ is interval function:

$$\hat{V}(b) = [V^-(b), V^+(b)]$$

2.2.1 Representation of value function

We represent the LB V^- by a set of vectors. The value at a belief b is a b's maximal projection onto the LB V^-. For the representation of UB V^+ we use a set of points. The UB V^+ is a lower convex hull of these points. So the value at a certain belief b is a projection of this point onto the lower convex hull. This projection is computed with a linear programming (LP). In bigger problems with higher dimensionality of the belief space, LP is able to calculate the projection much more efficiently than computing the hull explicitly [2].

2.2.2 Initialization of the bounds

HSVI needs some initial bounds. The initialization should take a imperceptible time next to time of running HSVI.

As an example of initialization of LB, we choose a strategy that always picks the first available action. We compute the expected rewards and get a vector that is now the only vector in the set that represents V^-.

To initialize the UB, we consider full observability and transform the problem into a MDP variant, then we solve it. Then we get UB values in the corners of the belief space. That is the initial set of points representing V^+ [9].

2.2.3 Local updating of lower and upper bound

Local updates are based on Bellman equation [2]. The value of taking action a in belief b is

$$Q^V(b,a) = \sum_s R(s,a)b(s) + \gamma \sum_o Pr(o \mid b,a)V(\tau(b,a,o)).$$

As you can see in the figure 2.2, the lower bound updating is done by adding a vector to the set V^- representing LB. Similarly The upper bound updating is done by adding a point to the set V^+ representing UB.

Also common procedure is to remove all the dominated vectors and points once in a while. Dominated in this context means that the vector or point has already no effect on the projecting of any belief onto the LB or UB [3].

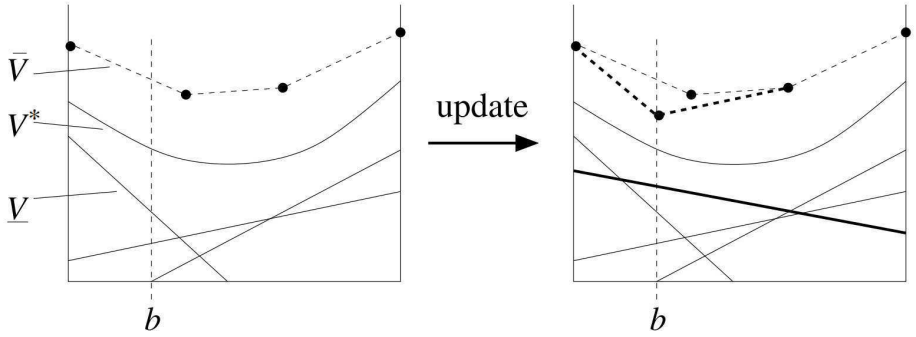


Figure 2.2: Locally updating at b [2]

2.2.4 Pseudocode [2]

Algorithm 1: $\pi \longleftarrow$ HSVI$(game, \epsilon)$

initialize lower bound V^-
initialize upper bound V^+
while $V^+(b_0) - V^-(b_0) > \epsilon$ **do**
  explore$(\epsilon, b_0, 0)$
end
return policy π according to the lower bound

Algorithm 2: explore(ϵ, b, t)

if $V^+(b) - V^-(b) < \epsilon\gamma^{-t}$ **then**
  return
end
select action a and observation o based on heuristics
explore$(\tau(b, a, o), \epsilon, t + 1)$
update local lower and upper bound at belief b

2.2.5 Anytime usage [2]

Sometimes we do not know a reasonable ϵ for the terminating condition. Sometimes we don't even want to set the ϵ. But we want to get from the algorithm the best strategy it can found in a given time. To achieve this we

can modify the HSVI algorithm as follows

Algorithm 3: $\pi \longleftarrow$ AnytimeHSVI($game, maxTime$)

initialize lower bound V^-
initialize upper bound V^+
while $runningTime < maxTime$ **do**
  explore($\eta(V^+(b_0) - V^-(b_0)), b_0, 0$)
end
return policy π according to the lower bound

where $0 < \eta < 1$.

Empirically, performance is not very sensitive to η. For example $\eta = 0.95$ is giving good performances (Smith, Simmons 2004).

2.3 Two-player One-sided Partially observable stochastic games (POSG) [1]

A two-player one-sided partially observable stochastic game G is a tuple $G = [S, A_1, A_2, O, T, R]$.
S ... set of states
A_1 ... set of actions of player 1
A_2 ... set of actions of player 2
O ... set of observations

The game can have an infinite time horizon. At each time t the game is in a state $s \in S$. Players pick their actions $a_1 \in A_1$ (player 1 (P1)) and $a_2 \in A_2$ (player 2 (P2)). They pick them concurrently. P1 gets an initial belief b_0, which is a probability distribution over the states S. Only P2 knows, in which state the game is.

The outcomes of continuing to time $t + 1$ are based on the picked actions - the game moves from state $s \in S$ to state $s' \in S$ and P1 obtains an observation $o \in O$ with probability $T_{s,a_1,a_2}(s', o)$. Also P1 receives a reward $r = R(s, a_1, a_2)$. Since we presume playing zero-sum game, P2 obtains a reward $-r$. Also we presume that discount factor $\gamma < 1$ is applied on the rewards. Players know the history of their played actions.

2.3.1 Nash equilibrium in non-cooperative games [3]

Nash equilibrium is a situation in a game where each player has a strategy, all players are supposed to know strategies of other players and no one can benefit by making any change in his own strategy.

Example

Let's consider a narrow street 1 mile long. There are Alice and Bob and both of them want to sell hot dogs in the street. Let's suppose that people would always buy hot dogs at the stand, that is the closest to their house. We can imagine the street as a horizontal line. The optimal solution, with considering how long distances people have to walk to the stands, would be that Alice would open her stand at 25% of the street (from the left) and Bob at 75% of the street. In this solution, both of them would serve hot dogs to half of the street. But Bob is smart and the only thing he cares about is money, so he moves to the 65% of the street. Now he operates $35 + \frac{65-25}{2}\% = 55\%$ of the street, meanwhile Alice operates only 45%. Alice observes this and and moves her stand closer to the center of the street. This continues and this way they both end up in the middle of the street next to each other each operating one half of the street and no one can move his stand and benefit from it. This is situation called Nash equilibrium.

2.3.2 Example of Two-player One-sided POSG - Simple pursuit-evasion game

In this game first player controls two agents - pursuers, the second player controls one agent - evader. In each step the agents move around an undirected graph, which is a grid having 3 rows and 6 columns. The goal of the pursuers is to catch the evader. When evader is caught, the game ends. The goal of the evader is to avoid the pursuers as long as possible. Every fourth step of the game the first player receives an observation about in which node of the graph the evader is located in the current step. Therefore states are tuples describing positions of all the three agents. Actions of the first player are combinations of possible moves of each individual pursuer according to the graph. Actions of the second player are available moves of the evader. There are 18 different observations - one for each node in the graph. Transition function simply takes the current state, the actions of both players and tells which state the player are going to get to with which probability and which observation the player one receives, if any. And the reward functions takes the same arguments and tells what reward the players get after playing such actions.

2.4 Heuristic Search Value Iteration for One-Sided POSGs (PG-HSVI) [1]

In this section we explain the PG-HSVI algorithm, which we modified within this thesis. This algorithm estimates a value of a one-sided partially observable stochastic game with infinite horizon. The estimate is done by restricting the horizon of the game and considering value functions of such games. The algorithm increases the horizon step by step in each of its iteration by gradual applying of a value backup operator H, which improves the value of the game estimation. Utilizing the value backup operator means both players to pick their strategies according to Nash equilibrium supposing that the previous iteration value function stands for the value of the following iteration.

2.4.1 Value Backup Operator [1]

To evaluate the value backup operator H in belief b (denoted $[HV](b)$) means to solve a phase of a game, in which both players pick their Nash equilibrium strategies. The rewards in $[Hv](b)$ are dependant on the discounted value of following iteration of the game and on the immediate reward. The immediate rewards are dependant only on the belief function about the current state of the game and actions played by both of the players:

$$R^{i}_{\pi_1,\pi_2} = \sum_{s\in S}\sum_{a_1\in A_1}\sum_{a_2\in A_2} b(s)\cdot\pi_1(a_1)\cdot\pi_2(s,a_2)\cdot R(s,a_1,a_2). \qquad (1)$$

The player 1 knows only the history of his actions and remembers observations he got during the game. He can benefit from these information by using them to calculate his belief for the next stage of the game:

$$b^{a_1,o}_{\pi_2}(s') = \frac{1}{Pr(o|a_1,\pi_2)}\sum_{s\in S}\sum_{a_2\in A_2} T_{s,a_1,a_2}(o,s')\cdot b(s)\cdot\pi_2(s,a_2). \qquad (2)$$

Then value of the following iteration of the game is an expectation computed over all pairs (a_1, o) from the values of a game that starts in a belief $b^{a_1,o}_{\pi_2}$:

$$R^{f}_{\pi_1,\pi_2}(v) = \sum_{a_1\in A_1}\sum_{o\in O}\pi_1(a)\cdot Pr(o|a_1,\pi_2)\cdot V(b^{a_1,o}_{\pi_2}). \qquad (3)$$

Given that the value functions are convex, reward from playing a strategy profile (π_1, π_2) is convex as well, if strategy π_1 is fixed and can be linear if strategy π_2 is fixed. Then the minimax theorem (von Neumann

1928 (PG-HSVI)) is applied and the Nash equilibrium strategy is solved by minimax/maximin:

$$[Hv](b) = \min_{\pi_2} \max_{\pi_1} \left(R^i_{\pi_1,\pi_2} + \gamma \cdot R^f_{\pi_1,\pi_2}(V) \right). \qquad (4)$$

2.4.2 Value Backup Operator Computation [1]

Value of a game is piece-wise linear convex function, therefore a set Γ of α-vectors can represent it. The α vector is an $|S|$-tuple describing affine value function v_{π_1} by defining the values of strategy π_1 in every pure belief (pure belief is a function that assigns 1 to only one state and 0 to all of the others). At first, the algorithm solves the problem from the context of the player 2, which picks his strategy π_2 with the intention of minimizing the reward of playing the best response by the player 1.

The value of playing π_2 against $a_1 \in A_1$ equals to immediate reward plus discounted following reward, and based on that a set of best-response constraints can be created:

$$V \geq \sum_{s \in S} \sum_{a_2 \in A_2} b(s) \cdot \pi_2(s, a_2) \cdot R(s, a_1, a_2) + \gamma \sum_{o \in O} Pr(o|a_1, \pi_2) \cdot v(b^{a_1,o}_{\pi_2}). \qquad (5)$$

Accepting that value of a game is described by a set Γ of α-vectors so that value $v(b) = \max_{\alpha \in \Gamma} \langle b, \alpha \rangle$, then it can be reformulated as a set of inequalities:

$$V(b^{a,o}_{\pi_2}) = \sum_{s' \in S} \alpha(s') \cdot b^{a,o}_{\pi_2}(s'); \quad \forall \alpha \in \Gamma. \qquad (6)$$

The term $Pr(o|a_1, \pi_2)$ in equations (2) and (5) cancels out, which results in creating the resulting linear program. This dual linear program is used to obtain the optimal strategy of the player 1. Equation (5) prescribes the strategy to play in the first iteration, while the equation (6) corresponds to the strategy to follow after observing (a, o) in the first iteration. Convergence of the value backup operator is proved in "Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games (Horak, Bosansky, Pechoucek 2017)".

2.4.3 The algorithm [1]

Estimation of the value function V is done by using a finite subset of first player's strategies. The approximation is done by keeping the lower and the upper bound in the same form as in the HSVI. The lower bound is represented by α-vectors so that the approximation is a maximal projection of

belief b onto the set Γ of α-vectors. Each one of the strategies is represented by one α-vector in Γ. The upper bound is a lower convex hull of points in a set Υ.

First step of the algorithm is the initialization of the lower and upper bounds. The lower bound is initialized by a simple strategy (usually a strategy, that repeats just a single one action). For the upper bound the algorithm solves an MDP version of the game. In other words it solves the same game, but it gives a full observability to player 1. Solving the MDP is a quick and easy way to get a reasonable upper bound of the game's value function.

Then it calls recursive function *explore* with belief b, ϵ and step t as arguments. The function uses the value backup operator to find the optimal strategy π_2 of player 2, that minimizes the maximal utility of player 1. Then it selects an action a_1 of player 1 and an observation o by searching for the maximal gap in the value function with the aim to reduce it. Then it computes a next belief $b^{a_1,o}_{\pi_2}$ after playing those actions and obtaining the observation. Then, if the gap in this next belief is bigger than ϵ, it calls recursively itself with the next belief as an argument instead of the current one. It continues until it gets into a certain belief so the value function's gap in it is lower than ϵ or it gets into such a distant step, that is discounted enough to be insignificant. After getting from the recursion it adds an α-vector corresponding to the strategy into the set Γ representing the lower bound, and a point into the set Υ representing the upper bound.

Since the lower bound is a maximal projection of the α-vectors in Γ, the updating by adding a vector can only result in either reducing a gap in a particular belief or in making no change, because there is no belief whose projection onto the α-vector would be maximal. The same logic can be applied to the upper bound. The initial call of the *explore* function is repeated, until the gap in the belief b_0 is lower that the desirable ϵ.

2.4.4 Pseudocode [1]

Algorithm 4: $v^* \longleftarrow \text{PG-HSVI}(\langle S, A_1, A_2, O, T, R\rangle, b_0, \gamma, \epsilon)$

initialize v
while $gap(v(b_0)) > \epsilon$ **do**
  explore$(b_0, \epsilon, 0)$
end
return v

Algorithm 5: explore(b, ϵ, t)

$\pi_2 \leftarrow$ optimal strategy of player 2 in $[Hv](b)$
$(a, o) \leftarrow$ select according to forward exploration heuristic
if $excess(v(b^{a_1,o}_{\pi_2}), t+1) > \epsilon$ **then**
  explore$(b^{a_1,o}_{\pi_2}, \epsilon, t+1)$
end
$\Gamma \leftarrow \Gamma \cup \{L\Gamma(b)\}$
$\Upsilon \leftarrow \Upsilon \cup \{U\Upsilon(b)\}$

Chapter 3

The Domain

In this part of the thesis we present the domain we chose for testing our variants of PG-HSVI. As a representative of POSGs we chose the Scotland Yard board game, because it is relatively famous, easy to understand and it corresponds with real-world scenarios. Police chasing a thief is a very common case. Also different instances of the game can be easily created. Furthermore the agents in the Scotland Yard have many available actions (because there are multiple units), which is great for testing, whether our contribution helps with solving games with big branching factor.

3.1 Scotland Yard [4]

Scotland yard in our case is a two-player game. There is player 1 (P1) having several agents (cops) and player 2 (P2) having just one agent (thief). In each iteration of the game each player moves his agent(s) along the edges of the graph. There are three types of transport - taxi, bus and underground. Cops have a certain amount of tickets for each type of them. The transport is not limited for the thief. P1 does not know, in which node the thief is, but after each iteration of the game he obtains an observation, which type of transport the thief used.

3.1.1 The goal of the game

P1 tries to catch thief by his agents as soon as possible. Catching thief means that in particular iteration one cop stays in the same node of the graph as the thief.

P2 tries thief to escape the cops.

3.1.2 Simplification

However, Scotland Yard game with all of its characteristics and rules induces a very large set of states and, consequently, large memory requirements. In a game, where graph is a grid 3x4, P1 has just two agents and both of them having total of 7 tickets for transport (that means the maximum length of the game is 7 steps), there are roughly 600.000 states, 15 million transitions and the file describing this game takes almost 1 GB of space. Therefore we decided to remove tickets from the game. After this step this game is not about whether cops can catch the thief, but for how long time is the thief able to escape them. This way the games are much smaller and we can focus on larger graphs.

3.1.3 Graph

Graph is a tuple <N, T, B, U>. N is a set of nodes. T, B, U are sets of edges. T represents a set of taxi edges. B represents a set of bus edges. U represents a set of underground edges.

3.1.4 State

Variant A of the game (with the tickets)

State is a tuple (A, t, B), where A is a vector of length n (number of agent of P1) consisting of nodes from N representing positions of agents of P1. t is a vector of length n consisting of vectors Xi of length 3. Xi represents number of tickets for each vehicle for agent i. B is a node from N representing position of the thief. Special states are s_1 and s_2. State s_1 is representing win of P1 (thief is caught). State s_2 is representing win of P2 (cops used already all of their tickets and can not move anymore).

Variant B of the game (without the tickets)

State is a tuple (A, B), where A is a vector of length n consisting of nodes from N representing positions of agents of P1, B is a node from N. Special state is S1 representing win of P1 (thief is caught). The set of all states is denoted S.

3.1.5 Moving around the graph

Agent of P2 moves in each iteration along one particular edge. P1 receives an observation about which type of transport the thief used.

Variant A

Move of agent of P1 is possible, if agent has a ticket for particular vehicle according to the type of the edge. Agent loses this ticket. In each iteration each agent must move along an edge in graph whenever it is possible. Also action of P1 is not correct, if after this action there exists a node with two cops in it.

Variant B

In each iteration each agent must move along an edge in graph whenever it is possible. Also action of P1 is not correct, if after this action there exists a node with two cops in it.

3.2 Generator

Before going through this section we need to mention *partitions*. In each time t the player 1 can be sure about a subset p of states S, that the game is definitely in one of the states from p and can not be in any other state. It is because the player 1 knows the position of his agents, but the term *state* includes the position of the thief. Practically in a partition are only the states, that share positions of police officers, but differs in the position of the thief. For instance a partition can be a set

$$p = \{(cops : (2,4), thief : 0), (cops : (2,4), thief : 1), (cops : (2,4), thief : 3)\}$$

in a game with 5 nodes and 2 police officers.

3.2.1 The goal of this task

The goal was to design and implement a program (generator), that satisfies these conditions:

1. Takes as an input text describing parameters of the variant of Scotland Yard game as described above.

   Example of such input:

   ```
   # number of nodes
   5
   # taxi edges (undirected graph)
   0 1, 0 2, 1 3, 1 4, 2 3, 3 4
   # bus edges
   ```

```
0 4
# underground edges
2 4
# number of agents of P1 (cops)
2
# initial locations of agents (first cops)
2 4 1
# discount factor
0.9500
```

2. The program relatively quickly generates input file for implemented PG-HSVI described above.

   The file consists of the following:

   - all possible states including information about a partition they belong to
   - all playable actions of P1 including information, which partitions they are playable in
   - all playable actions of P2 including information, which states they are playable in
   - enumeration of transitions

     $$state \times p1Action \times p2Action \rightarrow nextState \times observation \times probability$$

   - enumeration of rewards

     $$state \times p1Action \times p2Action \rightarrow reward$$

3.2.2 Symmetry of states

When designing the generator, we removed symmetry from the set of states. Symmetrical states are for example $state(cops : (1, 3, 4), thief : 5)$, $state(cops : (3, 1, 4), thief : 5)$ and $state(cops : (4, 1, 3), thief : 5)$ (numbers represent nodes). In other words, in this game it is irrelevant, which agent is in which node. Which nodes are at time t occupied by player 1 is the thing that matters. We decided to keep just states with positions of cops in ascending order. This way we reduced the sets of states to its $\frac{1}{n!}$ (n is a number of cops), because there are $n!$ permutations for each set of n positions and we keep only one of them. Therefore the amount of actions, transitions and rewards was reduced to their $\frac{1}{n!}$, because the for each of the states there would be different but symmetrical actions, transitions and rewards.

Chapter 4

Incremental strategy generation variant of PG-HSVI (ISG-PG-HSVI)

In this section we present the way we designed and implemented the incremental strategy generation method for PG-HSVI. By default the PG-HSVI takes a game as defined above, initializes the lower and the upper bounds and then, by recursive applying of value backup operator, reduces the gap between them until the gap is small enough. Because of the large amount of states and actions, the solving of the game by the algorithm can be relatively slow for bigger graphs. But the ISG-PG-HSVI takes a simplified version of the game, solves it more quickly and saves reusable data during computation. After that it takes a little less simplified version of the game and solves the new game with the help of precomputed data saved from previous iteration, which may help to solve the game more quickly or precisely. This cycle repeats until the full game is solved.

4.1 The simplification of the game

The goal of running the PG-HSVI on a simplified game is to get some results, that could be reusable in the next run on a less simplified game than the previous one. In this case the lower bound is the reusable result. Let us explain.

The algorithm takes a game description as an input. With this description it initializes a generator, which then generates the simplified games. The simplification lies in giving less available actions to player 1. In terms of the game, not all combinations of moves around the graph will be allowed for the police officers.

In fact, from the implementational point of view, even for the simplified games the generator generates all the possible actions. The *giving less actions to player 1* is done so that the generator chooses and adds actions to the

subset of all actions of player 1, and then, while generating the input file for the PG-HSVI algorithm, skips enumeration of all the transitions and rewards that contain an action of player 1, that is not in the mentioned subset.

The user must specify, how many actions will be available in each iteration, for instance 4 iterations with 30%, 50%, 70% and 100% of actions (denoted as portions of actions). The simplified game is then significantly smaller. For example, since in the input file there is significantly more transitions and rewards than states (for example in a full game with 18 states and 3 police officers there are 12.000 states, but over 400.000 transitions and rewards), decreasing the amount of available actions for player 1 (thus reducing the amount of transitions and rewards) by 70% reduces the size of the input file by 69%. In the experiments we test, among others, how large impact does it have on the performance of the algorithms.

The actions added to the subset are chosen randomly, but the used set of actions must remain in the next iteration, the generator can only add more actions into the set. The reason for it is that the lower bound of a game with less actions of player 1 could exceed the lower bound of a game with more, but different, actions of player 1. In other words, the less actions could be by chance the "better" actions than the actions in the larger set, which would result in a higher value of the smaller game. On the other hand, if the actions from the previous iteration stay and another actions become available, the lower bound can only rise. Therefore the next run of PG-HSVI on the new game can use the set of α-vectors from the previous run as an initial lower bound and does not have to start with the original initial lower bound, which was an only one α-vector representing a simple strategy.

The upper bound can not be reused. The value of the simplified game can be lower than the value of the full game and during the run of the algorithm the upper bound can get arbitrarily close to the value function. Reusing the upper bound in a next run of PG-HSVI could result in the value of the lower bound in any belief being higher that the value of upper bound in the same belief.

Similar logic can be applied for the reason why we can not make the game even smaller by giving less actions available to player 2 at the same time. Doing so could result in the value of the simplified game being higher than the value of the full game. Then, after giving all the available actions to player 2, the value of the game might drop and that could result again in the value of the lower bound in any belief being higher that the value of upper bound in the same belief.

For the purpose of implementing the incremental strategy generation method we had to modify the original implementation of the PG-HSVI algorithm. We added a possibility to save the lower bound to a file in the

end of the run of the algorithm, and a possibility to load the lower bound in the beginning of the next run and use it to solve the new game. This modification is in the following pseudocode denoted PG-HSVI*.

4.2 The algorithm

In the beginning the ISG-PG-HSVI takes as arguments a game description g, the portions of actions P, the initial belief b_0, the discount factor γ and the upper bound on the error ϵ. The game description g is a tuple $\langle N, E_t, E_b, E_u, \psi \rangle$, where N are vertices, E_t, E_b, E_u are taxi, bus and underground edges and ψ are the initial position of the agents. The P is an n-tuple of integers describing in percents how many action will be available for player 1.

Afterwards, the algorithm initializes the generator G with parameters g and P. Within the initialization the generator prepares for generating the games as an input files for the PG-HSVI*. Then the generator generates the first and the most simplified game. After that the PG-HSVI* is run, which solves the game the same way as the original PG-HSVI, but in the end it saves the lower bound into a file. Subsequently, a for cycle is run, which always lets the generator to generate a new game with more actions for player 1 according to P, and then runs the PG-HSVI*, which loads the saved lower bound and then saves the one it computed. The cycle continues until the full game is solved.

4.3 Pseudocode

Algorithm 6: $v^* \leftarrow$ ISG-PG-HSVI$(g, P, b_0, \gamma, \epsilon)$

initialize the generator G with parameters g, P
$\langle S, A_1, A_2, O, T, R \rangle, b_0 \leftarrow$ G.get_next_game()
$v^*, \Gamma \leftarrow$ PG-HSVI*$(\langle S, A_1, A_2, O, T, R \rangle, b_0, \gamma, \epsilon)$
for $i \leftarrow 0$ **to** $|P| - 1$ **do**
  $\langle S, A_1, A_2, O, T, R \rangle, b_0 \leftarrow$ G.get_next_game()
  $v^*, \Gamma \leftarrow$ PG-HSVI*$(\langle S, A_1, A_2, O, T, R \rangle, b_0, \gamma, \epsilon, \Gamma)$
end
return $v*$

4.4 Heuristic variant of ISG-PG-HSVI

In this section we present the way we designed and implemented a heuristic variant of ISG-PG-HSVI. We decided to design a domain independent heuristic, so it can be applied to other problems as well. The goal of this task was to show whether there exists a subset of actions of player 1 (small enough to make the algorithm solve the game considerably faster), for which the algorithm might give a solid approximation of the value of the game.

This variant of ISG-PG-HSVI has two parts. First, it lets the generator to generate many different games with the same amount of randomly chosen actions (the first portion of actions). It is clear that there are actions that can be more useful than others, for instance, in a game with a graph being a grid and the police officers standing about in the middle, an action in which the officers go in the same direction is probably more valuable than an action, where the officers go in different directions. It musts also remember the seeds that were used to pick the actions. After generating the games, it runs PG-HSVI* on each of them. Then it goes through the results and finds the game, where the subset of actions had the best results (the highest value of a game). Second, it runs ISG-PG-HSVI with exactly the same actions.

This variant takes the same parameters as the ISG-PG-HSVI, but additionally the user must specify the number n of such generated games and a time limit t for the running PG-HSVI* on them. The time limit is important because solving the game almost precisely is time consuming and the potential of the actions is mostly shown within the first iterations anyway.

The i^{th} element of sequence of portions of actions P is denoted P_i.

4.5 Pseudocode

Algorithm 7: $v^* \leftarrow$ heuristic ISG-PG-HSVI$(g, P, b_0, \gamma, \epsilon, n, t)$

initialize the generator G with parameters g, P
G.generate_games(n, P_1)
for $i \leftarrow 1$ **to** n **do**
  $v_i^* \leftarrow$ PG-HSVI$^*(\langle S, A_{1_i}, A_2, O, T_i, R_i \rangle, b_0, \gamma, \epsilon, t)$
end
$i \leftarrow argmax_i\ v_i^*(b_0)$
G.set_the_initial_actions(i)
$\langle S, A_1, A_2, O, T, R \rangle, b_0 \leftarrow$ G.get_next_game()
$v^*, \Gamma \leftarrow$ PG-HSVI$^*(\langle S, A_1, A_2, O, T, R \rangle, b_0, \gamma, \epsilon)$
for $i \leftarrow 0$ **to** $|P| - 1$ **do**
  $\langle S, A_1, A_2, O, T, R \rangle, b_0 \leftarrow$ G.get_next_game()
  $v^*, \Gamma \leftarrow$ PG-HSVI$^*(\langle S, A_1, A_2, O, T, R \rangle, b_0, \gamma, \epsilon, \Gamma)$
end
return $v*$

Chapter 5

Experiments

In the last chapter we describe the experimental results. There were two goals. The first goal was to determine the potential of ISG-PG-HSVI to solve problems faster that the original PG-HSVI. The second goal was to determine, whether there are subsets of actions of player 1 that influence the results significantly in a positive way for the player 1 compared to the majority of other subsets of actions. The reason for this goal is to find out, whether there is a potential for attempting to find a way to choose actions of player 1, that should be added to the game beforehand. That might be an opportunity for future work. Furthermore, in this section by the term *actions* we mean only the actions of player 1.

5.1 Tests details

5.1.1 The games

We designed 3 different games. We aimed to create diverse games with a goal to analyze the performance on different scenarios to fully understand the characteristics of the method.

The first game (denoted #12) has a graph that is a grid 3x4, therefore it has twelve nodes. All the grid edges are taxi edges, but there are three longer bus edges and one long diagonal edge. Furthermore there are two police officers.

The second game (denoted #18) is more complex. In the graph that are three cycles, each formed by 6 nodes, thus the graph has 18 nodes. Two of the cycles have only bus edges, while the third cycle has only taxi edges. Those cycles are randomly connected by 5 underground edges. Moreover there are three police officers.

The third game (denoted #24) has a graph consisting of 4 cycles

formed by 6 nodes, therefore there are 24 nodes in the graph. Two of the cycles have only taxi edges, while the other two cycles have only bus edges. These cycles are connected overall by 8 underground edges. Furthermore there are three police officers.

5.1.2 The parameters for the algorithms

All of the following parameters were common for both the ISG-PG-HSVI and the heuristic variant, unless stated otherwise. The were two different sequences of portions of actions P. The first one was $(30, 44, 64, 100)$ and the second one was $(60, 80, 100)$. The discount factor γ was always set to 0.95. The time limit was set to 2 hours of pure time for running the HSVI for all of the games. For the heuristic part the numbers n of the runs of the PG-HSVI* on the smallest variant of the game were 100 and 30 for the first game, 50 and 30 for the second game and 12 and 8 for the last game. The first number is always for the run starting with 30% of actions and the second numbers is always for the run starting with 60% of actions.

Both the ISG-PG-HSVI algorithm and the heuristic variant we run on each of the games with each of the sequences of portions of actions. We run the ISG-PG-HSVI several times with randomly chosen actions and present the average result. Furthermore we run the original PG-HSVI on the same game for comparison. All the tests were run on a single 2.20 GHz CPU. The largest tested problems required up to 31 GB of RAM.

Parameter	Value
Games	#12, #18, #24
P	$P^1 = (30, 44, 64, 100)$, $P^2 = (60, 80, 100)$
γ	0.95
Time limit	2 hours
Processor	1 CPU 2.20 GHz
RAM	64 GB

Table 5.1: Parameters and their values

5.2 The Results

In this section we present the results of the experiments. First we evaluate the results in terms of the speed of solving the problems. After that we show and analyze the results from the perspective of differences between the value functions of the same games with different actions available for player 1.

From the results we can say that the percentage of actions, for which the average value of the game is getting close to the value of the full game, differs very much between the games. In Figures 5.1 and 5.3 one can see that in the first game the player 1 needs at least roughly 80% of actions so his utility on average is less than 10 points lower than the value of the full game, but in the second game he need only 30% of actions so his utility on average is in the same distance from the value of the full game.

Further we denote the best found subset of actions by the heuristic variant of ISG-PG-HSVI containing $x\%$ of all available actions by A^*_x.

5.2.1 The fastness of the algorithms

For the comparison of the speed of the algorithms we decided to focus on how much time the algorithms need for the initialization and how much time they need to lower the difference between lower and upper bound in belief b_0 below a certain ϵ. In this section we skip the game #12, because time needed for solving the game is only a few seconds and the results would not be relevant.

In the following each table represents one configuration of one algorithm, for instance a running the ISG-PG-HSVI algorithm on a game #18 with $P = (60, 80, 100)$. The heuristic variant of ISG-PG-HSVI is in the following denoted HISG-PG-HSVI. Each column denoted "$x\%$" in the tables represents a single run of the PG-HSVI algorithm on a game with $x\%$ of actions. The row denoted *Start* shows how much time does an algorithm need for the initialization (reading the input file, initialization of all inner structures, lower bound, upper bound, etc.). The following rows denoted "Width $< x$" show how much time does an algorithm need for reducing the gap between lower and upper bound in belief b_0 below x. The unit of the values is a second. A "-" is written instead of the time, if the algorithm is not able to reduce the gap enough within the time limit.

The Game #18

The results of running the algorithms on this game can be seen in tables from 5.2 to 5.6. In the tables we can clearly see that the sizes of portions of actions have big influence of the time needed for initialization. For example the time needed for initialization of the ISG-PG-HSVI with 30% of actions is 178 seconds meanwhile with 100% of actions it is 436 seconds. Furthermore we can see in the tables, that in this game the ISG-PG-HSVI needs significantly more time for both the initialization and reducing the gap below the limit than HISG-PG-HSVI. That means that in this game the better actions have a positive impact on the time needed for computing.

Also if we look at the columns denoted 100%, one can see that in three out of four cases the getting some precomputed lower bound helps to initialize the algorithms faster. But overall the ISG-PG-HSVI and its heuristic variant are not faster the the original PG-HSVI, because of the time needed for the initialization. For example the overall time of the ISG-PG-HSVI with P^1 and $\epsilon = 1$ is $182+251+320+450 = 1203$, which is way more than 535. But in this game the average value of the game with only 30% of actions is roughly 70.2, while the value of the full game is 79.330 (Fig. 5.3). Then in this case if we want only a solid approximation of the value, we can stop the algorithm after solving the game with one of the smaller subsets of actions and the overall time needed for this would be significantly reduced. For example solving the game with 60% of action would take approximately 371 seconds, which is way less than 535 seconds.

time (s)	100%
Start	519
Width < 1	535

Table 5.2: The original PG-HSVI on the game #18

time (s)	30%	44%	64%	100%
Start	178	229	301	436
Width < 1	182	251	320	450

Table 5.3: ISG-PG-HSVI on the game #18 starting with 30% of actions

time (s)	30%	44%	64%	100%
Start	140	178	268	397
Width < 1	145	182	275	405

Table 5.4: HISG-PG-HSVI on the game #18 starting with 30% of actions

time (s)	60%	80%	100%
Start	339	415	527
Width < 1	371	439	540

Table 5.5: ISG-PG-HSVI on the game #18 starting with 60% of actions

time (s)	60%	80%	100%
Start	252	324	363
Width < 1	261	327	370

Table 5.6: HISG-PG-HSVI on the game #18 starting with 60% of actions

The Game #24

The results of running the algorithms on this game can be seen in tables from 5.7 to 5.11. Results say that solving a bigger game causes a significant increase of the time needed for the initialization. In this case the algorithms need for a graph having 33% more nodes roughly 10 times more time for the initialization. Moreover the average values of the games are not very good approximations of the value of the full game. The value starts to be relevant in games with 80% of actions, where the average lower bound value is 60.9, whereas the lower bound value of the full game is 70.3 (Fig. 5.5). But the results from running the heuristic variant show that even value of a game with 44% is a very good approximation of the lower bound value of the full game (lower bound values differ by 7 points). That means once a user has a subset A_x^* of actions, he can run the PG-HSVI on a game with this subset, which would result in a very good approximation and a short time of solving. Solving the game with 44% of actions would takes approximately 50 minutes, instead of 87 minutes for the full game.

time (s)	100%
Start	5154
Width < 10	5205
Width < 6	5239

Table 5.7: The original PG-HSVI on the game #24

time (s)	30%	44%	64%	100%
Start	2338	2953	3596	5241
Width < 10	2338	-	-	5320
Width < 6	2338	-	-	-

Table 5.8: ISG-PG-HSVI on the game #24 starting with 30% of actions

time (s)	30%	44%	64%	100%
Start	1732	2533	3602	5636
Width < 10	-	2918	4190	5752
Width < 6	-	-	-	-

Table 5.9: HISG-PG-HSVI on the game #24 starting with 30% of actions

time (s)	60%	80%	100%
Start	3429	4143	4888
Width < 10	-	4331	4897
Width < 6	-	-	4920

Table 5.10: ISG-PG-HSVI on the game #24 starting with 60% of actions

time (s)	60%	80%	100%
Start	3544	4572	5626
Width < 10	3713	4572	5626
Width < 6	3804	4653	5675

Table 5.11: HISG-PG-HSVI on the game #24 starting with 60% of actions

5.2.2 The divergence in values of the games between different subsets of actions

In this section we analyze the results from the perspective of divergence of the value functions of the same games with different actions available for player 1. In the following figures (from 5.1 to 5.6) we demonstrate the lower and upper bound values in b_0 that the algorithms are able to find in a given time. Each column denoted "s: x, p: y" represents one run of PG-HSVI algorithm on a game with y% of actions. The columns that have common "x" were all parts of one run of ISG-PG-HSVI, that started on a game with x% of actions. The first four columns represent the sequence of actions P^1, the next three columns represent the sequence of actions P^2 and the last column represents the value function computed by the original PG-HSVI.

Game #12

In the game #12 the value of the full game is 67.24. In the figures 5.1 and 5.2 we can see, that for the sequence of actions P^1, the average lower bound value for 64% of actions is roughly 40.8, but the the HISG-PG-HSVI was able to find a set A^* of actions, for which the value of the game was 62, which is a very solid approximation. As one can see in the figures, for all the portions of actions there are significant differences in the values for different subsets.

Game #18

In the game #18, as one can see in the figures 5.3 and 5.4, the average value of the game with only 30% of actions is 70.2, whereas the value of the full game is 79.3, which is actually a very good approximation for such a

small subset of actions. That is probably caused by that the graph by its size and shape provides a big advantage for the police officers and then they are able to catch the thief quickly even when the possibilities of their movements are very limited. Because of that the differences between the ISG-PG-HSVI and its heuristic variant are noticeable, but not significant.

Game #24

In the game #24 the value of the game is between 70.3 and 75.1. In the figures 5.5 and 5.6 we can see that on average the smaller subsets of actions are unable to approximate the value of the full game. Only with 80% of actions the average values are within a 10 point range far from it. But also the figures say that the value of the game with actions as a superset of A^*_{30} containing 44% of all available actions is between 63.4 and 72.9, which is a solid approximation for such a small subset. Furthermore the values of lower and upper bounds in b_0 of the game with actions A^*_{60} are almost identical to the values of lower and upper bounds of the full game.

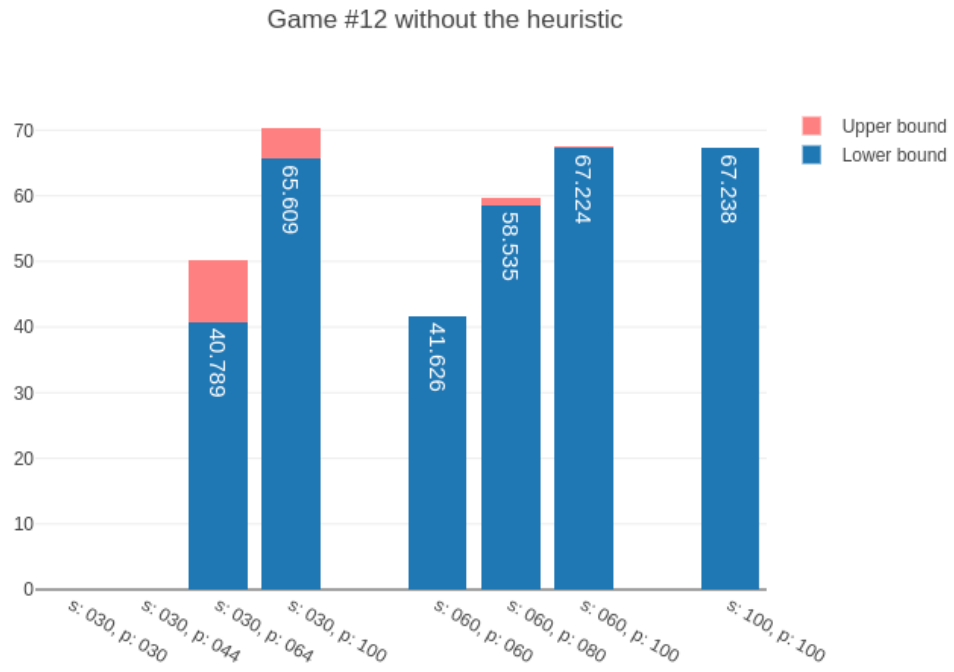


Figure 5.1: ISG-PG-HSVI - game #12

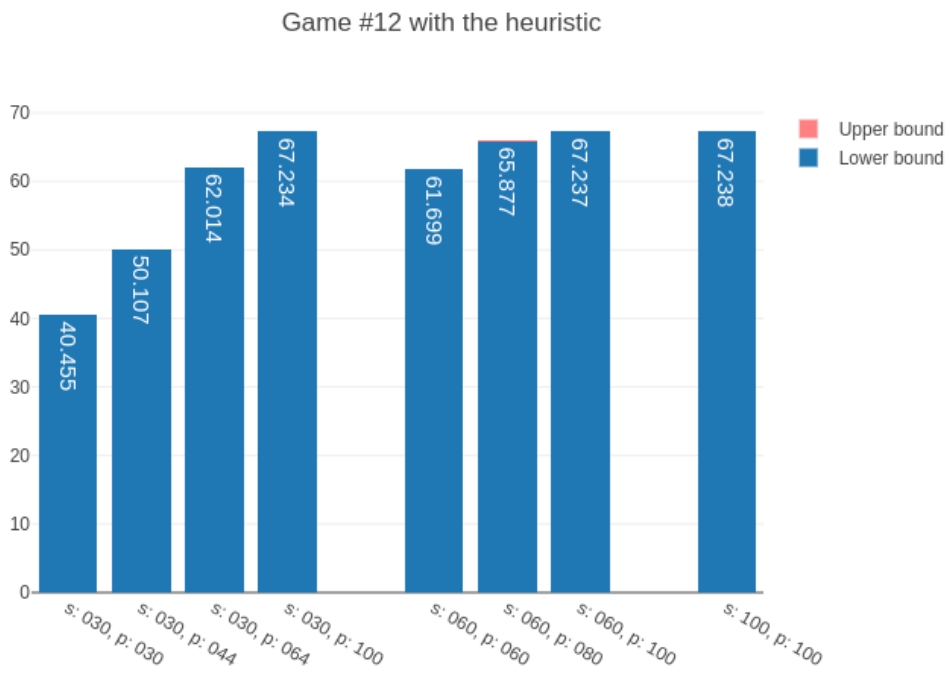


Figure 5.2: Heuristic ISG-PG-HSVI - game #12

5.2.3 Conclusion

Since the differences between the average values of the games and the values of the games with A_x^* as actions were huge, it is clear that there is a big potential for a future work, that might aim to find a way to efficiently pick subsets of actions for which the values of the games would approximate the values of the full games very precisely. From the perspective of time, the ISG-PG-HSVI can be hardly faster than the original PG-HSVI, but we show that there is a significant decrease in the time needed to solve the same game with less actions. That means finding a way to pick a better subsets of actions might cause a significant reduce of the time needed to approximate the value of the game with a solid precision.

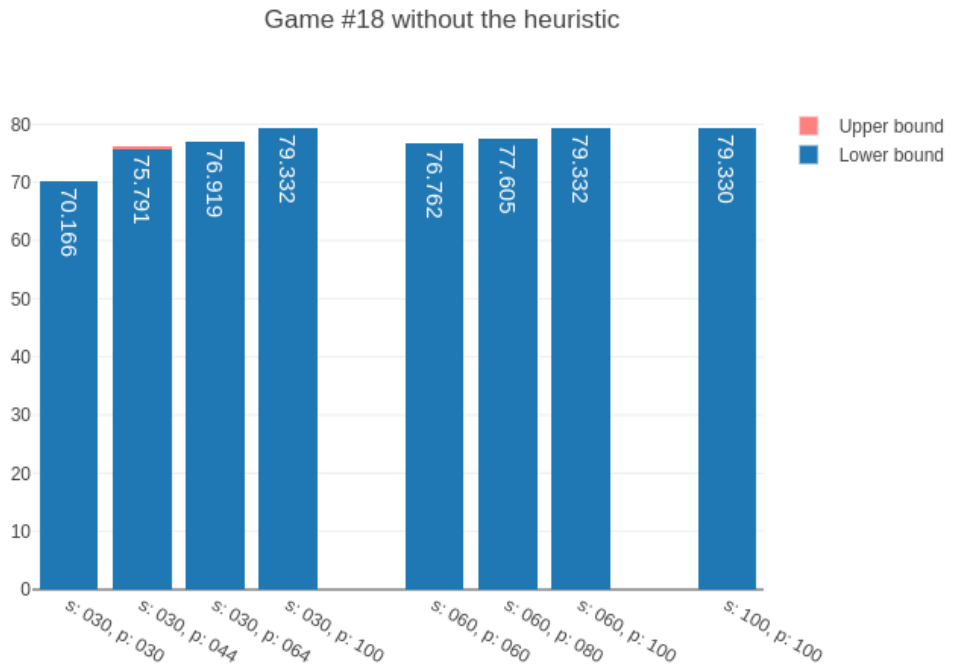


Figure 5.3: ISG-PG-HSVI - game #18

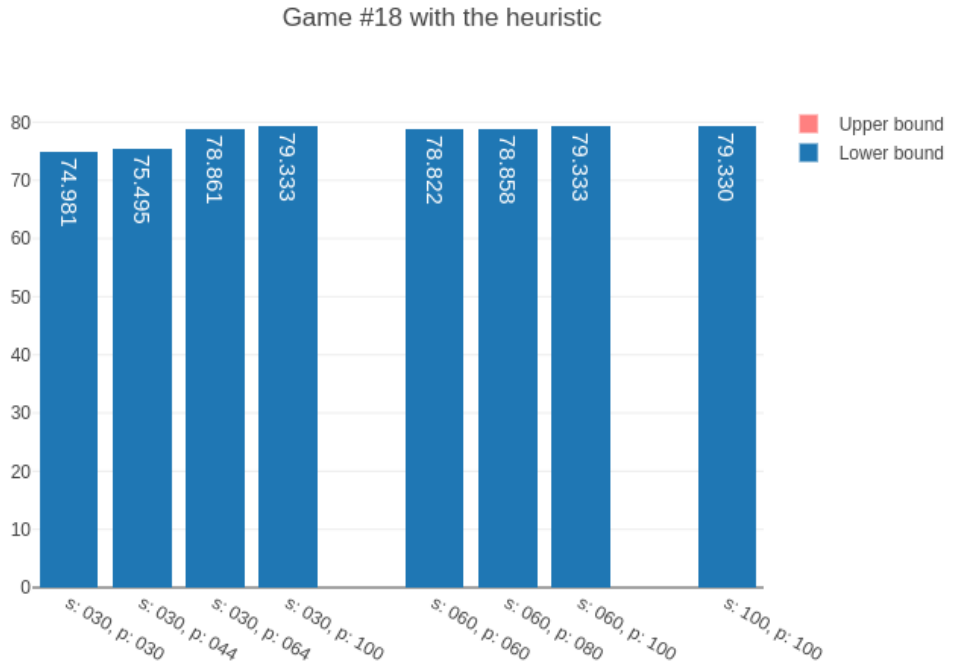


Figure 5.4: Heuristic ISG-PG-HSVI - game #18

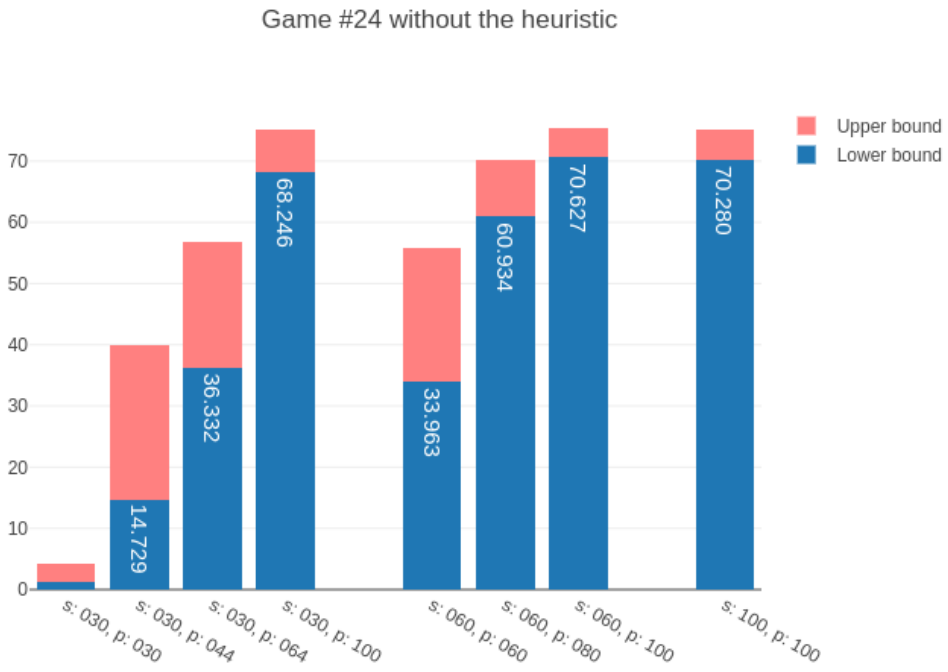


Figure 5.5: ISG-PG-HSVI - game #24

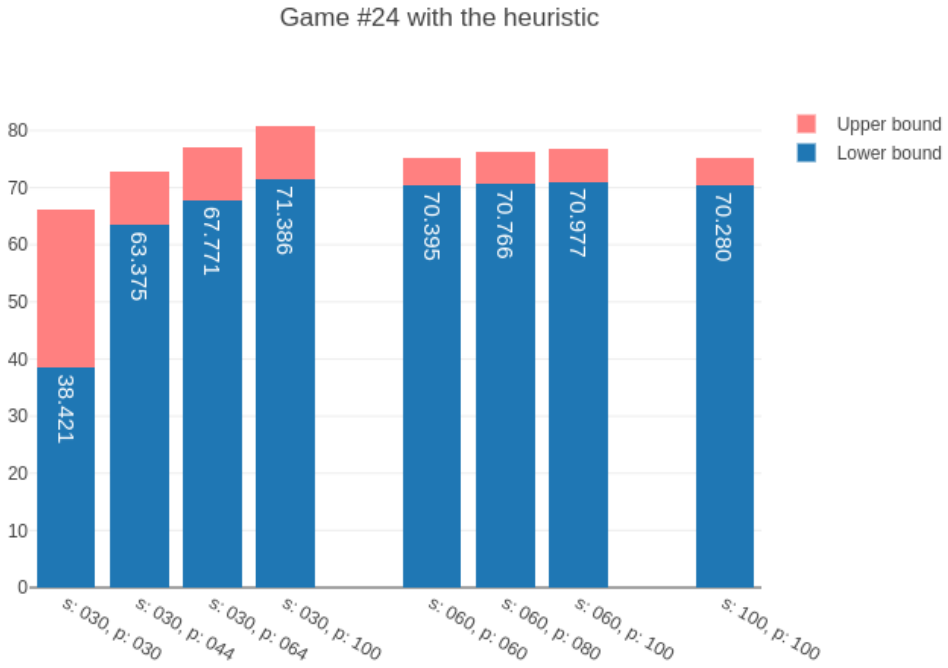


Figure 5.6: Heuristic ISG-PG-HSVI - game #24

Chapter 6

Conclusion and future work

In the thesis we focus on one-sided partially observable stochastic games, which represent many real world scenarios. We decided to choose a Scotland Yard board game as a representative of the POSGs.

There is a PG-HSVI algorithm, that can solve these games, but its scalability is limited. Therefore we decided to apply a incremental strategy generation method on the algorithm with the aim to improve the scalability, so that the algorithm would be able to solve games with more states and actions using less computation time. That means the new algorithm takes a game, excludes many actions of player 1 and runs the PG-HSVI. Then it gradually adds more actions into the game and lets the PG-HSVI to solve it again. This is repeated until the full game is solved.

We also designed and implemented a heuristic variant of this algorithm. The goal was to show, whether the chosen actions have a significant impact on the results.

The heuristic variant of the ISG-PG-HSVI runs the PG-HSVI many times on the game with a small subset of actions, then examines the results and runs the ISG-PG-HSVI on a game starting with the best set of actions found.

Finally, we performed experiments to compare the computation times of the introduced algorithms with the original PG-HSVI and to test the importance of the approach to choosing the actions. From the results we can say, that overall the ISG-PG-HSVI is not faster than the original algorithm, because the time needed for the initialization of the algorithm is considerable. But also the results demonstrate, that a single run of PG-HSVI on the same game with less action needs much less time for the whole process of computing.

Furthermore the results show, that the values of the games with different subsets of actions of player 1 of the same size differ vastly, for instance in one of the games there exists a subset containing only 30% of

all actions using which the heuristic variant of the ISG-PG-HSVI is able to approximate the value of the game very accurately saving 73% of the time compared to the original PG-HSVI. It means that there is a big potential for future work, which might try to design a strategy for choosing the actions added to the set of actions during the run of ISG-PG-HSVI. Together with the fact, that smaller games can be solved considerably faster, it might result in developing an algorithm, that would be able to approximate values of bigger games (in terms of number of states and actions) very precisely.

Bibliography

[1] K. Horák, B. Bošanskỳ, and M. Pěchouček, "Heuristic search value iteration for one-sided partially observable stochastic games," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[2] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004, pp. 520–527.

[3] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[4] R. AG. Scotland yard rules. [Online]. Available: https://www.ravensburger.us/spielanleitungen/ecm/Spielanleitungen/231250%20anl%201326321.pdf

[5] Partially observable markov decision processes lecture (fee ctu). [Online]. Available: https://cw.fel.cvut.cz/old/_media/courses/a4m36pah/pah-pomdp-v4.pdf

[6] N. M. A. Annapurna P. Patil, Bharath S., "Applications of game theory for cyber security system: A survey," *International Journal of Applied Engineering Research*.

[7] D. V. Janet Chen, Su-I Lu. Applications of game theory. [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/

[8] M. Hauskrecht, "Value-function approximations for partially observable markov decision processes," *Journal of artificial intelligence research*, vol. 13, pp. 33–94, 2000.

[9] K. J. Astrom, "Optimal control of markov processes with incomplete state information," *Journal of mathematical analysis and applications*, vol. 10, no. 1, pp. 174–205, 1965.

Appendix A

Source code

In this appendix we declare, which source code is created by us, which is modified by us and which is borrowed. In the project on the CD there are two folders - *generator* and *onesided-posgs*. Furthermore there are two more files - a brief readme.txt file describing how to run the introduced algorithms, and this thesis in a pdf format.

All the source code and documents in *generator* folder is created by us. This folder contains implementation of the generator, but also the introduced algorithms and scripts designed to run them and descriptions of several graphs for the generator.

The source code in the *onesided-posgs* folder is borrowed, but there are modifications done by us. In a file *main.cpp* we added some argument options and did some other minor changes. The biggest changes we made in files *hsvi.cpp* and *hsvi.h* as we implemented there several functions designed to save and load lower or upper bound of the value function, print results, etc. But all the necessary functions for running the original PG-HSVI were already implemented.