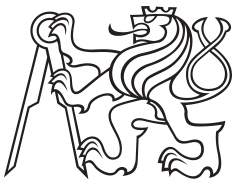
Bachelor Project



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Approximation of Bound Functions in Algorithms for Solving Stochastic Games

Jaroslav Šafář

Supervisor: Mgr. Branislav Bošanský, Ph.D.
Study program: Open Informatics
Branch of study: Computer and Informatic Science
May 2019

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Šafář Jaroslav** Personal ID number: **469852**

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:

Approximation of Bound Functions in Algorithms for Solving Stochastic Games

Bachelor's thesis title in Czech:

Aproximace konvexních funkcí v algoritmech pro řešení 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 (e.g., in security), however, the first recently developed algorithm PG-HSVI [1] for this class of games has insufficient scalability. One of the key steps of the algorithm is the approximation of the value function of the game using a lower-bound and an upper-bound function. These functions are represented as an upper envelope of linear functions and as a lower convex envelope of a set of points, respectively. Updates of these functions present one of the bottlenecks in the performance of the algorithm. The goal of the student is to:
1. Get familiar with the algorithm PG-HSVI.
2. Analyze possibilities for fast approximation of the convex functions representing these bound functions.
3. Select and implement at least two different methods for approximation of these functions.
4. Experimentally evaluate the impact of these changes compared to the original algorithm.

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] Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS), 22(4), 469-483.

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 would like to thank my thesis supervisor Mgr. Branislav Bošanský, Ph.D. for his patient guidance and helpful advises. The door to his office was always open whenever I had any question.

Also, I would like to express my gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study.

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, May 23, 2019

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 23. května 2019

Abstract

In this thesis, we focus on the approximation of the bound functions in the Heuristic Search Value Iteration (HSVI) algorithm for One-Sided Partially Observable Stochastic Games (OS-POSG). These are dynamic games with infinite horizon where only one player has imperfect information, and the opponent has full information. The bound functions approximate the value function of the game. The lower bound is represented as an upper envelope of linear functions, while the upper bound is represented as a lower convex envelope of a set of points. We focus only on the approximation of the upper bound mainly by using the Approximate Convex Hull algorithm. We show that the approximation of the upper bound is problematic and that for better results, it is necessary to focus on the approximation of the lower bound function as well.

Keywords: Game Theory, One-Sided Partially Observable Stochastic Games, Markov Decision Processes, Partially Observable Markov Decision Processes, Heuristic Search Value Iteration algorithm, Convex hull, Approximate convex hull

Supervisor: Mgr. Branislav Bošanský, Ph.D.
Artificial Intelligence Center,
Karlovo náměstí 13
12000 Praha 2

Abstrakt

V této práci se soustředíme na aproximaci konvexních funkcí v Heuristic Search Value Iteration algoritmu pro řešení Jednostranně Částečně Pozorovatelných Stochastických Her. Jedná se o dynamické hry, kde první hráč má neúplnou informaci o hře, zatímco druhý hráč má informaci úplnou. Konvexní funkce tvoří odhady tzv. value funkce celé hry. Dolní odhad je tvořen pomocí horní obálky lineárních funkcí, zatímco horní odhad je tvořen jako dolní konvexní obálka množiny bodů. V práci se zaměřujeme pouze na aproximaci horního odhadu převážně pomocí Aproximativního Convex Hull algoritmu. Ukazujeme, že aproximace horního odhadu je problematická a že pro lepší výsledky je zapotřebí se zaměřit také na aproximaci dolního odhadu.

Klíčová slova: Teorie her, Jednostranně Částečně Pozorovatelné Stochastické Hry, Markovovy Rozhodovací Procesy, Částečně Pozorovatelné Markovovy Rozhodovací Procesy, Heuristic Search Value Iteration algoritmus, Konvexní obal, Aproximativní konvexní obal

Překlad názvu: Aproximace konvexních funkcí v algoritmech pro řešení stochastických her

Contents

Figures

Tables

Chapter 1

Introduction

Game theory is the mathematical study of strategic interactions among the players. It is applied in many diverse fields such as economics, biology, psychology and, especially, computer science, where it can be used to model the real-world scenarios such as security and protection of critical objects. The security is a huge concern around the world these days, and limited resources often prevent full protection of critical objects at all times. Game theory can be useful in such scenarios. This was shown, for example, in the application of a game theoretic model for security at the Los Angeles International Airport [PJM+08], where it helped establish a security system around the whole airport, or in optimal resource allocations in a security of transportation systems, computer networks, and other critical infrastructure [KJT+09].

In the real-world security scenarios, the position of an attacker is usually unknown until they are discovered. It is said that the defender has partial observability or imperfect information. On the other hand, the attacker knowing everything is the worst possible case. Typically, the defender needs to protect the critical objects for a very long, undefined, time.

These scenarios can be modeled as Two-Player One-Sided Partially Observable Stochastic Games with the infinite horizon (OS-POSGs) where only one player (the defender) has imperfect information about the game as opposed to his opponent (the attacker), who has full information about the game. The expected outcome of the game is represented by a so-called value function which returns the expected reward of the first player. This value can be both positive (the defender stops the attacked) or negative (the attacker succeeds).

The goal in such games is to find the optimal strategy for the first player, which maximizes his expected reward. Finding such a strategy can often be computationally challenging.

One-Sided POSGs have been used in the past in specific domains such as patrolling games [VATS14], where patrolling units have imperfect information about the game, or pursuit-evasion games [HB16], where only the evader knows the positions of the pursuing units. In 2014 the first domain-independent Heuristic Search Value Iteration algorithm for One-Sided Partially Observable Stochastic Games [HBP17] was introduced. It is a generalization of the Heuristic Search Value Iteration algorithm (HSVI) for Partially Observable Markov Decision Processes (POMDPs) [SS04a], which deals with only one agent. The algorithm approximates the value function of the game with lower and upper bound convex functions. The lower bound is represented by a set of vectors representing hyperplanes, and the upper bound is represented by a lower convex envelope of a set of points. Unfortunately, the HSVI algorithm for POSGs has insufficient scalability in bigger games.

The focus of this thesis is to analyze possible approximations of the upper bound convex function to potentially improve the scalability of the algorithm. We will start with the following chapter by describing Partially Observable Markov Decision Processes and the original HSVI algorithm for POMDPs, which is the bases for the HSVI algorithm for POSGs. Two-Player One-Sided Partially Observable Stochastic Games are the main topic of the third chapter, together with a detailed description of the HSVI algorithm for POSGs. The next chapter will focus on the limitations of the HSVI algorithm for POSGs and some possible adjustments. The main modification of the upper bound function will focus on Approximate Convex Hull algorithm, which removes the points from the upper bound that can be well approximated by the rest of them. Last two chapters will be focused on experimental results and the conclusion.

Chapter 2

Heuristic Search Value Iteration for Partially Observable Markov Decision Processes

This chapter introduces HSVI algorithm for POMDPs, which is the basis for the HSVI algorithm for POSGs. The first section introduces Markov Decision Processes which are necessary to understand Partially Observable Markov Decision Processes described in the second section. The last section is focused on describing the HSVI algorithm for POMDPs itself.

2.1 Short introduction to Markov Decision Processes

Markov Decision Processes [RN09] model a single agent making decisions in a stochastic environment. It is assumed that the environment is **fully observable**, i.e., the agent always knows in which state he is. The goal for the agent starting from the initial state is to reach one of the goal states. If the environment were deterministic, a solution would consist of a sequence of actions that would lead the agent to one of the goal states. Unfortunately, the environment is stochastic; i.e., each action has only a certain probability of achieving the intended effect. Therefore a different approach must be introduced.

2.1.1 Definition of Markov Decision Processes

Definition 2.1. Markov Decision process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the stochastic transition function such that $\mathcal{T}(s'|a, s) = Pr\left[s_{t+1} = s' | a_t = a, s_t = s\right]$ and $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward function.

The transition function $\mathcal{T}(s'|a, s)$ determines the probability of reaching the state s' from the state s by action a. These probabilities depend only on the current state s and not on the earlier states visited by the agent. It is said that the transitions are **Markovian**. In each state s, the agent receives a reward $\mathcal{R}(s)$, which can be both positive or negative.

Denote $[s_0, s_1, s_2, \ldots]$ the sequence of states $s_i \in \mathcal{S}$ visited by the agent. If the sequence is infinite we talk about MDP with an **infinite horizon**. Standard way of assigning utility to the agent visiting the sequence of states is by **discounted rewards**:

$$U([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \tag{2.1}$$

where $\gamma \in (0, 1)$ is called the **discount factor**. The discount factor describes the preference of an agent for current rewards over future rewards.
Suppose that the rewards are bounded by $\mathcal{R}_{max}$, i.e. for all states $s \in \mathcal{S}$ $|\mathcal{R}(s)| \leq \mathcal{R}_{max}$. Then $U([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t) \leq \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{max} = \mathcal{R}_{max} \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma} \mathcal{R}_{max}$, due to the sum of an infinite geometric series.
Therefore, the utility of an infinite sequence of states is finite.

As it was already stated the solution to MDP cannot be a fixed sequence of actions because the agent might possibly end up in a different state than the desired goal. The solution to MDP is so called **policy** π which determines which action the agent should play at any state. The recommended action for the state s is denoted by $\pi(s)$. The quality of a given policy can be measured by the **expected utility**:

Definition 2.2. The expected utility obtained by an agent starting in the initial state $s \in \mathcal{S}$ and reaching the state $s_t \in \mathcal{S}$ at time t (assuming $s_0 = s$), while executing policy π, is given by

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t)\right] \tag{2.2}$$

where the expectation is with respect to the probability distribution over state sequences determined by s and π.

Naturally, an **optimal policy** π_s^*, which does not have to be unique, is the policy that yields the highest expected utility for an agent starting from the state $s \in \mathcal{S}$ and following the policy π, formally:

$$\pi_s^* = \arg\max_{\pi} V^{\pi}(s). \tag{2.3}$$

The consequence of discounted utilities with infinite horizon is that the optimal policy is independent of the starting state [RN09]. Therefore, we can write π^* for an optimal policy and $V^*(s) = V^{\pi^*}(s)$ as the **utility** or the **value of the state** s.

Assuming the utility $V^*(s)$ of any reachable state $s \in \mathcal{S}$ is known, the optimal policy π^* depends only on the current state s. The agent can choose an action that maximizes the expected utility of the subsequent state by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|a, s) V^*(s'). \tag{2.4}$$

where $\mathcal{A}(s)$ is a set of actions available to the agent in the state s.

2.1.2 Value Iteration

From the definition of the utility of the state s as the expected sum of discounted rewards from that point onward while executing the optimal policy (Equation 2.2) follows a relationship between the utility of a state s and the utility of its neighbors:

$$V^*(s) = \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|a, s) V^*(s'). \tag{2.5}$$

This equation is called the **Bellman equation**. It says that the utility of a state s is the sum of the immediate reward $\mathcal{R}(s)$ for that state and the expected discounted utility of the next state, assuming that the agent follows the optimal policy π^* and therefore chooses the optimal action $\pi^*(s)$.

Suppose that there are n possible states. For every single one of them, there is one Bellman equation. Therefore, there are n Bellman equations with n unknown utilities of the states. To find these unknown utilities, it is necessary to solve this system of equations, which are unfortunately nonlinear.

One way to solve this system of nonlinear equations is an iterative approach called **The value iteration algorithm** [RN09]:

1. Let $U_i(s)$ be the utility value for state s at the i-th iteration.
   Set the initial values for the utilities $U_0(s)$ to arbitrary values.

2. Calculate the right-hand side of every Bellman equation and use these new-found utilities for next iteration, formally:

$$U_{i+1}(s) \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|a, s) U(s'). \tag{2.6}$$

   This iteration step is called a **Bellman update**.

3. Repeat the previous step until the change in the values between the iterations is smaller than the desired precision.

The Value iteration algorithm is formally described in the (Algorithm 1).

Algorithm 1: Value Iteration for MDP

Result: Optimal utility function V^*
Input : MDP, desired precision ε
Output : V^*

for $s \in \mathcal{S}$ **do**
    $V'(s) \leftarrow 0$
repeat
    $V \leftarrow V'$
    $\delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
        $V'(s) \leftarrow \mathcal{R}(s) + \gamma \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{T}(s'|a, s) V(s')$
        if $|V'(s) - V(s)| > \delta$ **then**
            $\delta \leftarrow |V'(s) - V(s)|$
until $\delta < \frac{1-\gamma}{\gamma}\varepsilon$;
return V

It can be proven that the Value iteration algorithm converges. The proof can be found in [RN09]. The corresponding policy obtained by (Equation 2.4) is therefore optimal.

2.2 Partially Observable Markov Decision Processes

In previous section Markov Decision Processes assumed that the environment was fully observable, i.e., the agent always knew in which state he was. On the other hand, Partially Observable Markov Decision Processes

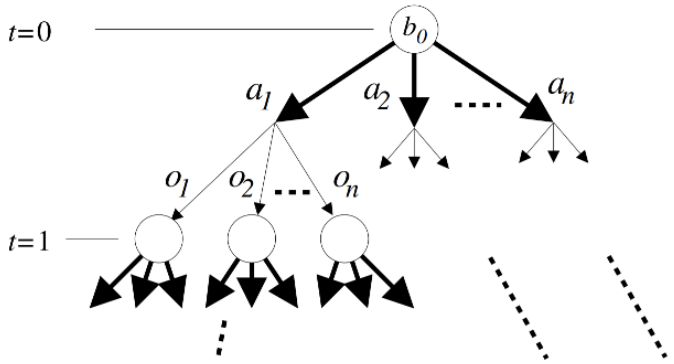


Figure 2.1: POMDP tree structure. **Source**: [SS04a]

[RN09] [SS04a] model a single agent acting under uncertainty in a **partially observable** environment, where the agent does not know the current state of the game. Therefore, he cannot execute the action $\pi(s)$ recommended for that state.

2.2.1 Definition of Partially Observable Markov Decision Processes

Because the agent does not directly observe the environment's state, he only knows the probabilities of being in a certain state. These probabilities are called **belief states**. The initial probability distribution is called the **initial belief**. The effects of the actions are again stochastic, i.e. the actions have only a certain probability to achieve the intended effect. At any belief state, the agent takes an action. Then he receives a reward and, unlike in MDP, a noisy observation. After that he moves to a new belief state. Formally POMDPs can be defined as follows:

Definition 2.3. Partially Observable Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, O, \mathcal{R}, \gamma, b_0 \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{O}$ is the set of observations, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the stochastic transition function such that $\mathcal{T}(s'|a, s) = Pr\,[s_{t+1} = s'|a_t = a, s_t = s]$, $O : \mathcal{O} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the stochastic observation function such that $O(o|a, s) = Pr[o_t = t|a_t = a, s_{t+1} = s]$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma < 1$ is the discount factor and b_0 is the initial belief where $b_0(s) = Pr[s_0 = s]$.

Suppose the agent selects an action a at belief point b, receives a noisy observation o and moves to a new belief b'. This process can naturally be viewed as a tree structure (Figure 2.1). Nodes of the tree represent beliefs where the agent must make a decision. The root of the tree corresponds

to the initial belief b_0. The directed edges starting from the node labeled with the belief b corresponds to the available actions. These edges branch to several others based on the observations that the agent can receive after choosing that action.

Denote $b(s)$ the actual probability of being in the state s given by the belief state b. Assume that the agent knows the history of his actions $\mathbf{a}^t = \{a_0, a_1, \ldots, a_t\}$ and history of his observations $\mathbf{o}^t = \{o_0, o_1, \ldots, o_t\}$ up to time t. Using this histories and the initial belief b_0 the agent can recursively calculate its current belief state at time $t+1$, denoted as

$$b_{t+1} = \tau(b_t, a_t, o_t) \tag{2.7}$$

If b was the previous belief state where the agent chose an action a and received an observation o, then the new belief state $b' = \tau(b, a, o)$ is given by

$$b'(s') = \eta O(o|a, s') \sum_{s \in \mathcal{S}} \mathcal{T}(s'|a, s) b(s), \tag{2.8}$$

where η is a normalizing constant that makes the belief state sum to 1.

The fundamental insight is that the optimal action, which should be performed at any time, depends only on the agent's current belief state. This optimal action is specified by a **policy** π which maps the current belief state b to a recommended action $\pi(b)$.

The quality of a policy π starting from a belief b can be measured by the **expected utility**:

Definition 2.4. The expected utility obtained by an agent following a policy π and starting from a belief b is defined as

$$V^{\pi}(b) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]. \tag{2.9}$$

The goal is to find an **optimal policy** π^* which maximizes the expected utility for an agent starting from the belief state b_0:

$$\pi^* = \arg\max_{\pi} V^{\pi}(b_0) \tag{2.10}$$

Example 2.5. Consider POMDP with two states. The agent has two actions available: either to stay in the current state, or move to the other one. These actions have 80% chance to succeed. The observations do not depend on the performed action, they only help determine the current state, correctly with probability of 70%. Formally:

$\mathcal{S} = \{s_1, s_2\}$, $\mathcal{A} = \{stay, go\}$, $\mathcal{O} = \{o_1, o_2\}$, $\mathcal{T}(s_i|stay, s_i) = 0.8$,
$\mathcal{T}(s_i|go, s_i) = 0.2$, $O(o_i|go, s_i) = O(o_i|stay, s_i) = 0.7$, $\forall i \in \{1, 2\}$.
$\mathcal{R}(s_1) = 0$, $\mathcal{R}(s_2) = 1$, $\gamma = 0.95$.

The following graph is a visualization of this example:

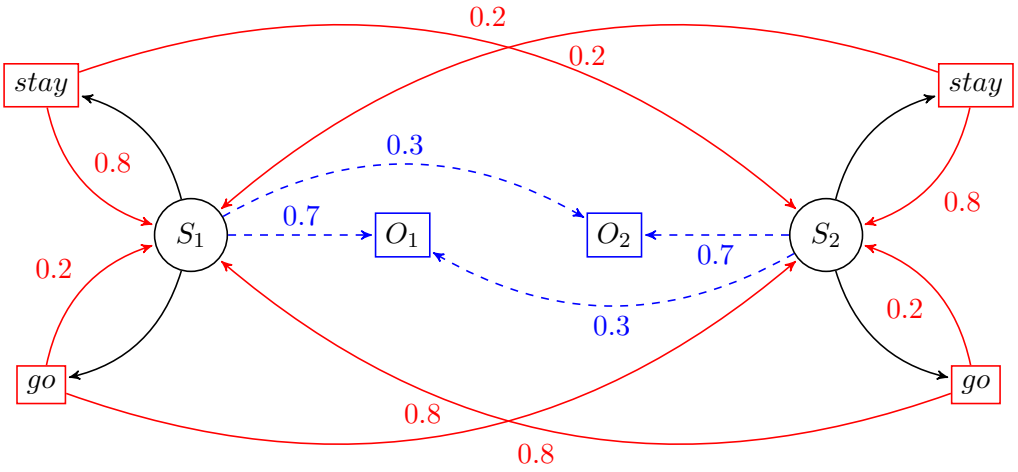


Figure 2.2: Graph of POMDP with two states

2.2.2 Value iteration for POMDPs

In MDP the value iteration algorithm founds utility for each one of the finite number of states. In POMDPs, there are infinitely many belief states representing any possible probability distribution over the set of states. Therefore, a similar approach is not possible.

Consider a fixed optimal policy π^* which generates an action $\pi^*(b)$ in a specific belief state b. Then the belief state is updated, and the process repeats. The policy is equivalent to a so-called **conditional plan** dependent on future observations.

Example 2.6. Consider the same POMDP as in Example 2.5. An example of a plan of length 2 could be: [*stay*, if $O = o_1$ then *go* else *stay*].

Let $\alpha_p(s)$ be the utility of a fixed plan p starting from a state s. Then the utility of the same plan p executed from the belief state b is only a sum of utilities $\alpha_p(s)$ weighted by the probabilities $b(s)$:

$$\sum_{s \in \mathcal{S}} \alpha_p(s) b(s) = \langle \alpha_p, b \rangle \tag{2.11}$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. From this expression follows that the expected utility of a fixed conditional plan p is linear in belief state b and corresponds to a hyperplane.

The optimal policy π^* can now be expressed in terms of plans as the policy which will choose to execute the plan with highest expected utility at any belief state b:

$$V^*(b) = V^{\pi^*}(b) = \max_p \langle \alpha_p, b \rangle. \tag{2.12}$$

It is reasonable to expect that there will be a very similar utility and identical policy in belief states which are close to a certain belief state b. These observations lead to the following statement: The utility function $V^*(b)$ is piecewise linear and convex in the belief b.

Consider step plans of depth 1. These plans would receive the reward for the current state plus the discounted reward for the state reached after the action. Once these utilities are known, the utilities for plans of depth 2 can be computed by considering each possible first action, each possible subsequent observation, and then each way of choosing a plan of depth 1 to execute for each observation. In general, for plan p of depth d with initial action a and a subplan of depth $d-1$ belonging to an observation o denoted as p_o, holds:

$$\alpha_p(s) = \mathcal{R}(s) + \gamma \left(\sum_{s' \in \mathcal{S}} \mathcal{T}(s'|a, s) \sum_{o \in \mathcal{O}} O(o|a, s') \alpha_{p_o}(s') \right). \tag{2.13}$$

This recursive equation is a foundation of a value iteration algorithm which can be found in [RN09].

Value iteration algorithm is computationally intractable for large state spaces, therefore, the approximate solutions are usually used. The usual goal of the approximate solution is to minimize the **regret** of the returned policy π for the initial belief b_0, which is defined as

$$regret(\pi, b_0) = V^*(b_0) - V^{\pi}(b_0) \tag{2.14}$$

2.3 Heuristic Search Value Iteration for POMDPs

Heuristic Search Value Iteration for Partially Observable Markov Decision Processes [SS04a] is an approximate algorithm that approximates the optimal

value function V^* by lower and upper bounds which are denoted as $\underline{V}$ and $\bar{V}$, respectively. Interval function $\hat{V}$ refers to them collectively:

$$\hat{V}(b) = \left[\underline{V}(b), \bar{V}(b)\right] \tag{2.15}$$

$$width(\hat{V}(b)) = \bar{V}(b) - \underline{V}(b) \tag{2.16}$$

The HSVI makes a local update at a specific belief, which is chosen by a forward explore in the search tree using heuristic that selects optimal actions and observations. The goal of the algorithm is to find a policy π such that $regret(\pi, b_0) \leq \varepsilon$ for the desired precision ε.

2.3.1 Value Function Representation

From (Equation 2.12) in the previous section follows that value function V^* can be represented by a (possibly infinite) set of vectors. In real-world applications, it is, of course, impossible to work with an infinite set of vectors, but for the discounted infinite-horizon case, a finite set can approximate V^* arbitrarily close.

This finite vector set formulation is used for the representation of the **lower bound** function $\underline{V}$ as the finite set Γ of α-vectors. The value at a belief point b can be calculated as:

$$\underline{V}(b) = \max_{\alpha \in \Gamma} \langle \alpha, b \rangle \tag{2.17}$$

With this representation it is easy to perform local update on the vector set by adding a new vector.

For the **upper bound**, the representation by a finite set Υ of belief/value points $(b_i, \bar{v}_i)$ is used. Before describing the representation itself, the basic terminology of **convex sets** is required. This terminology will also be used frequently in the following chapters.

Definition 2.7. The set $X \subseteq \mathbb{R}^n$ is called convex if for all $x \in X$, $y \in X$ and $\alpha \in \langle 0, 1 \rangle$, the $\alpha x + (1 - \alpha)y \in X$.

Definition 2.8. The convex combination of points $x_1, ..., x_k \in \mathbb{R}^n$ is the linear combination $\sum_{i=1}^{k} \alpha_i x_i$ such that $\sum_{i=1}^{k} \alpha_i = 1$ and $\alpha_i \geq 0 \ \ \forall i \in \{1, 2, ..., k\}$

Definition 2.9. The convex envelope (hull) of a finite set of points X is the set of all convex combinations of its points.

Having defined the convex hull of points, the value at belief point b is calculated by a linear program as the projection of b onto the convex hull of Υ. The local update is performed by adding a new point to the set.

2.3.2 Initialization of the HSVI Algorithm

The initialization of the **lower bound** is done by a blind policy method: consider π_a to be the policy of always selecting an action a. The lower bound $\underline{R}_a$ on the long-term reward of policy π_a can be calculated by assuming that the action a is always chosen in the worst possible state, formally:

$$\underline{R}_a = \sum_{t=0}^{\infty} \gamma^t \min_{s \in \mathcal{S}} \mathcal{R}(s,a) = \frac{1}{1-\gamma} \min_{s \in \mathcal{S}} \mathcal{R}(s,a) \tag{2.18}$$

Let

$$\underline{R} = \max_{a \in \mathcal{A}} \underline{R}_a. \tag{2.19}$$

Then the lower bound $\underline{V}$ is initialized by a single α-vector such that $\alpha(s) = \underline{R}$.

The initialization of the **upper bound** is done by assuming full observability and solving the MDP version of the problem. This solution provides upper bound values at the corners of the belief simplex. These points initialize the upper bound function $\bar{V}$.

2.3.3 Local Updates

The **Bellman equation** says that the value of choosing an action a in a belief b is

$$Q^V(b,a) = \sum_{s \in \mathcal{S}} R(s,a) b(s) + \gamma \sum_{o \in \mathcal{O}} Pr\left[o|b,a\right] V(\tau(b,a,o)) \tag{2.20}$$

The **Bellman update** H is the fundamental operation of value iteration. It is defined as follows:

Definition 2.10. The Bellman update H is defined as

$$HV(b) = \max_{a \in \mathcal{A}} Q^V(b,a) \tag{2.21}$$

HSVI uses **local update** at belief b based on operator H. The lower bound vector set Γ is updated by adding a vector which is result of **Backup**

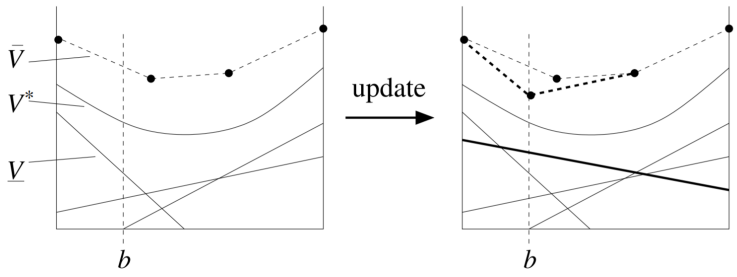


Figure 2.3: Local update of bound functions. **Source**: [SS04a]

Algorithm described in (Algorithm 2). The upper bound point set Υ is updated by adding a point $(b, H\bar{V}(b))$. Formally:

$$\Gamma \leftarrow \Gamma \cup backup(\underline{V}, b) \tag{2.22}$$

$$\Upsilon \leftarrow \Upsilon \cup (b, H\bar{V}(b)) \tag{2.23}$$

Algorithm 2: Backup Algorithm

Result: New vector to update Γ
Input : Lower bound $\underline{V}$, belief b
Output : vector β

for $a \in \mathcal{A}$ *and* $o \in \mathcal{O}$ **do**
  $\beta_{a,o} \leftarrow \arg\max_{\alpha \in \Gamma} \langle \alpha, \tau(b, a, o) \rangle$
for $a \in \mathcal{A}$ **do**
  $\beta_a(s) \leftarrow R(s, a) + \gamma \sum_{o \in \mathcal{O}} \sum_{s' \in \mathcal{S}} \beta_{a,o}(s') O(o|a, s') T(s'|a, s)$
$\beta \leftarrow \arg\max_{\beta_a} \langle \beta_a, b \rangle$
return β

The (Figure 2.3) shows the process of locally updating the lower and upper bound functions in belief b.

2.3.4 Forward Exploration Heuristic

The HSVI algorithm needs to find the belief points which contribute to the insufficient approximation of the value function V^* in the initial belief. Following **heuristic** provides a guideline for choosing an optimal action a^* and observation o^* in a belief b while searching the POMDP tree for such belief points. The resulting child node to visit will be $\tau(b, a^*, o^*)$. This heuristic needs to ensure that the Bellman update H at the chosen child $\tau(b, a^*, o^*)$ will reduce the uncertainty $width(\hat{V}(b))$ at the root b.

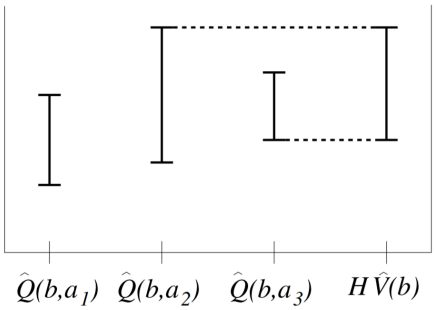


Figure 2.4: Relationship between $\hat{Q}(b, a_i)$ and $H\hat{V}(b)$. **Source**: [SS04a]

1. **Choosing an action**
Define the interval functions $\hat{Q}$ and $H\hat{V}(b)$ as follows:

$$\hat{Q}(b,a) = \left[\underline{Q^V}(b,a), Q^{\bar{V}}(b,a)\right] \tag{2.24}$$

$$H\hat{V}(b) = \left[H\underline{V}(b), H\bar{V}(b)\right] \tag{2.25}$$

The (Figure 2.4) shows the relationship between the bounds $\hat{Q}(b, a)$ on each potential action and the bounds $H\hat{V}(b)$ at belief b after the Bellman update.

From the definition of Bellman update H (Definition 2.10) follows that only the two $\hat{Q}(b, a)$ intervals, one with the maximal upper bound and the other with maximal lower bound, determine the $H\hat{V}(b)$ interval. Therefore, one of these actions should be pursued. But witch one? The action with greatest upper bound is better because if a sub-optimal action a^* were chosen, then the upper bound of a^* would eventually drop below the upper bound of another action. The similar thing does not work for the lower bound. Therefore, the optimal action can be found by:

$$a^* = \arg\max_{a \in \mathcal{A}} Q^{\bar{V}(b,a)} \tag{2.26}$$

2. **Choosing an observation**
For a fixed optimal action a^*, consider the relationship between $\hat{Q}(b, a^*)$ and the bounds at the child nodes $\tau(b, a^*, o)$ for any observation o. From the Bellman equation (Equation 2.20) follows that

$$width(\hat{Q}(b,a^*)) = \gamma \sum_{o \in \mathcal{O}} Pr\left[o|b,a\right] width(\tau(b,a,o)) \tag{2.27}$$

From this follows that the uncertainty $width(\hat{V}(b))$ at a belief b after an update is at most γ times a weighted average of its child nodes uncertainties and that the termination criterion for this search with the desired precision ε will be $width(\hat{V}(b)) \leq \varepsilon\gamma^{-t}$.

Definition 2.11. Let ε be the desired precision. The excess uncertainty at belief b in depth t is defined as

$$excess(b,t) = width(\hat{V}(b)) - \varepsilon\gamma^{-t} \tag{2.28}$$

The node with negative excess uncertainty satisfies the termination condition. It holds that the excess uncertainty at belief b is at most a probability-weighted sum of the excess uncertainties at its childern:

$$excess(b,t) \leq \sum_{o\in\mathcal{O}} Pr\left[o|b,a^*\right] excess(\tau(b,a^*,o),t+1). \tag{2.29}$$

Therefore, the child that contributes the most to the excess uncertainty at b gives the optimal observation:

$$o^* = \arg\max_{o\in\mathcal{O}} \left(Pr\left[o|b,a^*\right] excess(\tau(b,a^*,o),t+1)\right). \tag{2.30}$$

2.3.5 Summary and convergence of the HSVI Algorithm

The HSVI algorithm takes desired precision ε and the initial belief point b_0 and returns a policy π such that $regret(\pi,b_0) \leq \varepsilon$.

It stores the upper and lower bounds on the optimal value function and locally updates them at specific beliefs which are chosen by exploring forward in the search tree according to a heuristic that selects optimal actions and observations. The forward exploration heuristic and local update are summarized in the following **Explore Algorithm** (Algorithm 3).

Algorithm 3: Explore Algorithm: $explore(b,\varepsilon,t)$

Result: Updates the bound functions in specific beliefs which are found by forward exploration
Input : the root belief b, desired precision ε, depth t of belief b
Output : Updated sets Γ and Υ

```
if width(V̄(b)) ≤ εγ^{-t} then
    return
else
    a* ← arg max_{a∈A} Q^{V̄}(b,a)
    o* ← arg max_{o∈O} [Pr(o|b,a*) excess(τ(b,a*,o),t+1)]
    explore(τ(b,a*,o*),ε,t+1)
    Γ ← Γ ∪ backup(V̲,b)
    Υ ← Υ ∪ (b, HV̄(b))
```

The complete **HSVI Algorithm for POMDPs** is described in (Algorithm 4).

Algorithm 4: HSVI for POMPDs

Result: Policy π such that $regret(\pi, b_o) \leq \varepsilon$
Input : POMDP, desired precision ε, initial belief b_0
Output : Policy π

Initialize the bounds $\hat{V}$.
while $width(\hat{V}(b_0)) > \varepsilon$ **do**
    $explore(b_0, \varepsilon, 0)$
Having achieved the desired precision, return the policy π corresponding to the lower bound $\underline{V}$.

The following theorem provides some of the most important theoretical results. Full theoretical discussion is presented in [SS04a].

Theorem 2.12. *For the HSVI Algorithm it holds that:*

- *The $regret(\pi, b_0)$ of the policy π returned by HSVI Algorithm is at most ε.*
- *There is a finite depth t_{max} such that all nodes with depth $t \geq t_{max}$ have negative excess uncertainty and therefore satisfy the termination condition for the explore algorithm. This finite depth is equal to*

$$t_{max} = \lceil \log_{\gamma}(\frac{\varepsilon}{\|\bar{V_0} - \underline{V_0}\|_{\infty}}) \rceil \tag{2.31}$$

  where $\bar{V_0}$ and $\underline{V_0}$ are the initial bound functions.
- *HSVI Algorithm is guaranteed to terminate after performing at most u_{max} updates, where*

$$u_{max} = t_{max} \frac{(|\mathcal{A}||\mathcal{O}|)^{t_{max}+1} - 1}{|\mathcal{A}||\mathcal{O}| - 1} \tag{2.32}$$

Chapter 3

Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games

The previous chapter discussed theory and algorithms for a single agent acting in a stochastic environment. The presented HSVI algorithm for POMDPs is the basis for the HSVI algorithm for POSGs described in this chapter. The first section focuses on basic terminology of Two-Player One-Sided Partially Observable Stochastic Games, which generalize POMDPs. The second section describes Value Iteration algorithm for POSGs, which unfortunately cannot scale for bigger problems. The last section finally presents the HSVI algorithm for POSGs.

3.1 Two-Player One-Sided Partially Observable Stochastic Games

The presence of the second player in a stochastic environment is a generalization of POMDPs from the previous chapter. Such environment with two players, where only the first player has imperfect information about the environment, is called **Two-Player One-Sided Partially Observable Stochastic Game** [HBP17].

3.1.1 Definition of Two-Player One-Sided Partially Observable Stochastic Games

Definition 3.1. Two-Player One-Sided Partially Observable Stochastic Game G is a tuple $G = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{O}, \mathcal{T}, \mathcal{R}, \gamma, b_0 \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}_1$ is the set of actions of the first player, $\mathcal{A}_2$ is the set of actions of the second player, $\mathcal{O}$ is the set of observations, $\mathcal{T} : \mathcal{O} \times \mathcal{S} \times \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \to \mathbb{R}$ is the transition function such that $\mathcal{T}(o, s'|s, a_1, a_2) = Pr\left[o^t = o, s^{t+1} = s'|s^t = s, a_1^t = a_1, a_2^t = a_2\right]$, $\mathcal{R} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \to \mathbb{R}$ is the reward function, $\gamma < 1$ is the discount factor and $b_0 \in \Delta(\mathcal{S})$ is the initial belief where $b_0(s) = Pr\left[s_0 = s\right]$.

The game is played for infinite number of **stages**. At each stage the game is in one of the states $s \in \mathcal{S}$ where players choose their actions $a_1 \in \mathcal{A}_1$ and $a_2 \in \mathcal{A}_2$ to be played. Initial state $b_0 \in \Delta(\mathcal{S})$ is a probability distribution over the set of states $\mathcal{S}$.

After both players played their respective actions, the first player, who has imperfect information about the game, receives an observation $o \in \mathcal{O}$, the game moves to a state $s' \in \mathcal{S}$ with probability $\mathcal{T}(o, s'|s, a_1, a_2)$ and the first player receives a reward $\mathcal{R}(s, a_1, a_2)$. It is assumed that the game is **zero-sum** game, i.e. the second player receives $-\mathcal{R}(s, a_1, a_2)$. The rewards are again discounted over time with discount factor $\gamma < 1$. Players do not observe their rewards during the game.

Perfect recall is also assumed, therefore, each player remembers everything they did in the past. A history of the first player with imperfect information is limited to a set $(\mathcal{A}_1 \times \mathcal{O})^t$. The second player has full information about the game, therefore $\mathcal{S} \times (\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{O} \times \mathcal{S})^t$ is a set of her history.

The **strategies** σ_1, σ_2 of both players are mappings from the set of their respective histories to the set of their respective actions.

Example 3.2. Consider an environment defined as the following simple graph with only two vertices joined by an edge:

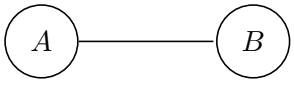


Figure 3.1: Simple graph environment

This environment defines the following OS-POSG with two players, who can move between the vertices:

States of this game are tuples representing the position of each player: $\mathcal{S} = \{(A, A), (A, B), (B, A), (B, B)\}$. The actions for both players are either to *stay* in the current vertex or to *move* to the other one: $\mathcal{A}_1 = \mathcal{A}_2 = \{stay, go\}$. The first player has imperfect information about the game whereas the second player has full information. The observations for the first player determines whether he sees the second player in the same vertex or not: $\mathcal{O} = \{yes, no\}$. The first player repeatedly tries to catch the second player. This situation is represented by the states where both players are in the same vertex. Every time the game gets into such a state, the first player receives a reward of 1. On the contrary, for states where both players are in the different vertices, he receives a reward of -1. The game is zero-sum. Therefore, the second player receives a reward -1 for being caught and 1 otherwise. This rewards also demonstrates the fact that the goal for the second player is to avoid the first player as much as possible. Let us assume the discounted rewards by a factor $\gamma = 0.95$ and the initial belief for the first player $b_0 = (0.20, 0.30, 0.40, 0.10)$. The transition function has 5 parameters. To define the whole transition probability function it would require to define all $4 \cdot 2 \cdot 4 \cdot 2 \cdot 2 = 128$ possible transitions.
To demonstrate transitions for one stage of the game, suppose that the game is in a state (A, A) and both players chose the action *go*: $a_1 = a_2 = go$. Then the transitions of this stage of the game can be graphically demonstrated as follows:

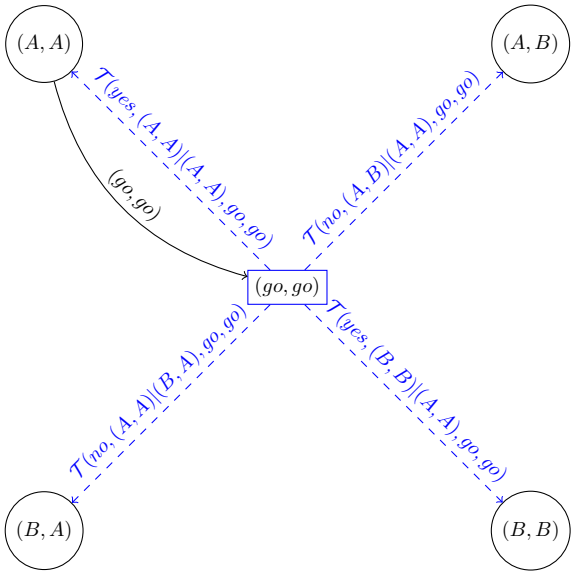


Figure 3.2: Transitions of one stage of the game

3.1.2 Value of Strategy and Value of the Game

The **value** of strategy σ_1 of the first player is the expected reward of the first player playing σ_1 while the opponent plays her best response, formally:

Definition 3.3. The value of the strategy σ_1 of the first player is a function $V_{\sigma_1} : \Delta(\mathcal{S}) \to \mathbb{R}$ which, given the initial belief $b_0 \in \Delta(\mathcal{S})$, returns the expected utility $V_{\sigma_1}(b_0)$ of the first player playing σ_1 and the second player her best-response.

The value of the game G, or the **value function**, represents the expected outcome of the game. It is the value of the best strategy available for each of the initial beliefs $b_0 \in \Delta(\mathcal{S})$, formally:

Definition 3.4. The value function of the game G is a function $V^* : \Delta(\mathcal{S}) \to \mathbb{R}$ which, given the initial belief $b_0 \in \Delta(\mathcal{S})$, returns the value of the best strategy of the first player for that initial belief, i.e. $V^*(b_0) = \sup_{\sigma_1} V_{\sigma_1}(b_0)$.

Definition 3.5. The pure belief of the state s is defined as

$$b_s(s') = \begin{cases} 1, & s = s' \\ 0, & s \neq s' \end{cases} \tag{3.1}$$

Consider a value of a fixed strategy σ_1 of the first player evaluated in the pure belief b_s of the state s. Denote the vector of such values as α_{σ_1}, i.e. $\alpha_{\sigma_1}(s) = V_{\sigma_1}(b_s)$. Then the following lemma holds

Lemma 3.6 (Lemma 1 in [HBP17]). *Let α_{σ_1} be the vector corresponding to a fixed strategy σ_1 of the first player defined as above. Then the value V_{σ_1} of strategy σ_1 is linear in the initial belief and it holds that*

$$V_{\sigma_1}(b_0) = \sum_{s \in \mathcal{S}} \alpha_{\sigma_1}(s) b_0(s) = \langle \alpha_{\sigma_1}, b_0 \rangle \tag{3.2}$$

Proof. The proof relies on the fact that the second player knows the initial state of the game. Therefore, the value of a fixed strategy σ_1 is sum of values of vector α_{σ_1}, weighted by probabilities b_0. □

Example 3.7. Consider the game from (Example 3.2) with the initial belief $b_0 = (0.20, 0.30, 0.40, 0.10)$ and the vector $\alpha_{\sigma_1} = (-1, 0, 1, 2)$ representing values of the fixed strategy σ_1 in the pure beliefs. Then value of strategy σ_1 executed from the initial belief b_0 is

$$V_{\sigma_1}(b_0) = \langle \alpha_{\sigma_1}, b_0 \rangle = -1 \cdot 0.20 + 0 \cdot 0.30 + 1 \cdot 0.40 + 2 \cdot 0.10 = 0.4.$$

Definition 3.8. A real function f is called K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathcal{D}(f) : |f(x_1) - f(x_2)| \leq K\|x_1 - x_2\|$.

Denote

$$L = \min_{(s,a_1,a_2)} \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s,a_1,a_2) \qquad U = \max_{(s,a_1,a_2)} \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s,a_1,a_2) \tag{3.3}$$

then the following lemma holds.

Lemma 3.9 (Lemma 2 in [HBP17]). *Value function V_{σ_1} of a fixed strategy σ_1 of the first player is $(U-L)$-Lipschitz.*

Theorem 3.10 (Theorem 1 in [HBP17]). *The value function V^* of the game G is convex in the initial belief and $(U-L)$-Lipschitz.*

Proof. From the definition of the value function V^* (Definition 3.4) follows that V^* is supremum of a set of $(U-L)$-Lipschitz functions (Lemma 3.9). Supremum taken over the set of bounded $(U-L)$-Lipschitz functions is also $(U-L)$-Lipschitz function. These functions are also linear (Lemma 3.6), therefore, supremum over the set of linear functions is convex function. □

3.2 Value Iteration Algorithm for POSGs

The **Value Iteration Algorithm** [HBP17] for solving one-sided POSGs approximates the value function V^* of the game G with infinite horizon by value functions of the same game with finite horizon. Each iteration of the algorithm improves the approximation by increasing the horizon using the **value backup operator** H.

3.2.1 Value Backup Operator

One iteration of the algorithm can be represented by evaluating **value backup operator** H at belief point b, which is denoted as $[HV](b)$. This evaluation corresponds to solving one stage of the game: players choose their Nash equilibrium strategies while assuming that the value of the subsequent game is represented by the value function from the previous iteration. Denote the strategy of the first player in the current stage as $\pi_1 \in \Delta(\mathcal{A}_1)$ and the strategy of the second player in the current stage as $\pi_2 : \mathcal{S} \to \Delta(\mathcal{A}_2)$.

The utilities in the current stage depend both on immediate rewards $\mathcal{R}$ and on the discounted value of subsequent game represented by value function V.

The **immediate reward** depends only on actions played by the players:

$$\mathcal{R}^{imm}_{\pi_1,\pi_2} = \sum_{s\in\mathcal{S}} \sum_{a_1\in\mathcal{A}_1} \sum_{a_2\in\mathcal{A}_2} b(s)\cdot\pi_1(a_1)\cdot\pi_2(s,a_2)\cdot\mathcal{R}(s,a_1,a_2) \tag{3.4}$$

After both players played their respective actions, the first player needs to update his **belief for the subsequent game** using an action $a_1 \in \mathcal{A}_1$ he played and an observation $o \in \mathcal{O}$ he received:

$$b^{a,o}_{\pi_2}(s') = \frac{1}{Pr\left[o|a_1,\pi_2\right]} \sum_{s\in\mathcal{S}} \sum_{a_2\in\mathcal{A}_2} \mathcal{T}(s'|o,s,a_1,a_2)\cdot b(s)\cdot\pi_2(s,a_2) \tag{3.5}$$

The **value of the subsequent game** is the expectation taken over actions and observations of the first player from the values of a game starting in belief $b^{a,o}_{\pi_2}$:

$$\mathcal{R}^{subs}_{\pi_1,\pi_2}(V) = \sum_{a_1\in\mathcal{A}_1} \sum_{o\in\mathcal{O}} \pi_1(a_1)\cdot Pr\left[o|a_1,\pi_2\right]\cdot V(b^{a,o}_{\pi_2}) \tag{3.6}$$

Then the Nash equilibrium strategy is solved by Minimax theorem [SLB08], which represents the **Bellman equation** for one-sided POSGs:

$$[HV](b) = \min_{\pi_2}\max_{\pi_1}(\mathcal{R}^{imm}_{\pi_1,\pi_2} + \gamma\mathcal{R}^{subs}_{\pi_1,\pi_2}(V)) \tag{3.7}$$

3.2.2 Computation of Value Backup Operator

Consider a strategy σ_1 of the first player. From (Lemma 3.6) follows that the value V_{σ_1} of such strategy can be represented by an α-vector such that for any belief point b it holds that $V_{\sigma_1}(b) = \langle\alpha, b\rangle$.

If the value function V of the game G is piecewise linear and convex (PWLC) it can be represented by a set Γ of α-vectors corresponding to the value functions of the fixed strategies V_{σ_1}. Therefore, the value function V evaluated at any belief point b is

$$V(b) = \max_{\alpha\in\Gamma}\langle\alpha, b\rangle \tag{3.8}$$

The value backup $[HV](b)$ can now be evaluated using linear programming.

Strategy of the Second Player

In the current stage represented by the value backup $[HV](b)$ the second player needs to choose her strategy π_2 to minimize the utility V of the first player who plays his best response $a_1 \in \mathcal{A}_1$. The value of playing strategy π_2 against an action $a_1 \in \mathcal{A}_1$ is

$$\mathcal{R}^{imm}_{a_1,\pi_2} + \gamma \mathcal{R}^{succ}_{a_1,\pi_2}(v) \tag{3.9}$$

From this equation, a set of best-response constrains can be constructed, one for each action a_1:

$$v \geq \sum_{s \in \mathcal{S}} \sum_{a_2 \in \mathcal{A}_2} b(s) \cdot \pi_2(s, a_2) \cdot \mathcal{R}(s, a_1, a_2) + \gamma \sum_{o \in \mathcal{O}} Pr\left[o | a_1, \pi_2\right] \cdot V(b^{a,o}_{\pi_2}) \tag{3.10}$$

If the value function V is represented by a set Γ of α-vectors such that $V(b) = \max_{\alpha \in \Gamma} \langle \alpha, b \rangle$, then $\forall \alpha \in \Gamma$:

$$V(b^{a,o}_{\pi_2}) \geq \sum_{s' \in \mathcal{S}} \alpha(s') \cdot b^{a,o}_{\pi_2}(s') \tag{3.11}$$

where $b^{a,o}_{\pi_2}(s')$ is represented by linear constraints in (Equation 3.5).

Strategy of the First Player

As it was already mentioned the value function V can be approximated by a PWLC function represented by a finite set Γ of α-vectors which correspond to a finite subset of linear value functions of the fixed strategies of the first player.

The dual linear program is used to find the optimal strategy of the first player. Duals of (Equation 3.10) corresponds to the strategy to play in the first stage when the history of the first player is empty. Duals of (Equation 3.11) corresponds to the strategy to follow when (a, o) was observed in the first stage.

3.2.3 Convergence of the Value Backup Operator

The last thing to show is that a repetitive application of the value backup operator H always converges to the value function V^*. The convergence

can be shown by proving that the value backup operator H is a contraction mapping with a factor $\gamma < 1$.

Lemma 3.11 (Lemma 3 in [HBP17]). *Let V, V' be value functions, $b \in \Delta(\mathcal{S})$ be a belief point and π_1, π_2 (resp. π_1', π_2') be Nash equilibrial strategies in the stage $[HV](b)$ (resp. $[HV'](b)$). Assume that for every action-observation pair (a, o) of the first player, $|V(b^{a,o}_{\pi_2}) - V'(b^{a,o}_{\pi_2})| \leq \mu$. Then $|[HV](b) - [HV'](b)| \leq \gamma\mu$, where $\gamma < 1$.*

Theorem 3.12 (Theorem 2 in [HBP17]). *The value backup operator H is a contraction mapping under the norm $\|V - V'\| = \max_{b \in \Delta(\mathcal{S})} |V(b) - V'(b)|$. Therefore, it has a unique fixpoint: the value function of the infinite horizon game.*

Proof. Let $\|V - V'\| \leq \mu$. Then for every $b^{a,o}_{\pi_2}$ from (Lemma 3.11) follows that $|V(b^{a,o}_{\pi_2}) - V'(b^{a,o}_{\pi_2})| \leq \mu$ and for every belief b it holds that $|[HV](b) - [HV'](b)| \leq \gamma\mu$. From the Banach's fixed point theorem [K.C07] follows the uniqueness of the fixpoint and the convergence of the algorithm. □

3.3 Heuristic Search Value Iteration Algorithm for POSGs

Similarly to POMDPs, the value iteration algorithm cannot scale for practical problems with a bigger set of states. This point based **Heuristic Search Value Iteration Algorithm** [HBP17] is a generalization of HSVI algorithm for POMDPs described in the previous chapter. Therefore, many parts of the algorithm will be the same or similar.

The HSVI algorithm bounds and approximates the true value function V^* by a pair of PWLC functions: **lower bound** $\underline{V}$ and **upper bound** $\bar{V}$. The lower bound $\underline{V}$ is represented by finite set Γ of α-vectors and the upper bound $\bar{V}$ is represented as a lower convex envelope of a set Υ of points. The interval function $\hat{V}$ refers to both bound functions collectively:

$$\hat{V}(b) = \left[\underline{V}(b), \bar{V}(b)\right] \tag{3.12}$$

$$gap(\hat{V}(b)) = \bar{V}(b) - \underline{V}(b) \tag{3.13}$$

The goal of the algorithm is to find these functions $\hat{V}$ such that $gap(\hat{V}(b_0)) \leq \varepsilon$, where ε is the desired precision. The algorithm alters the functions $\hat{V}$ by adding new elements to their sets. It is done by **point-based updates** of operator H at a belief point b which is selected by **froward exploration heuristic**. These selected belief points contribute to the fact that the $gap(\hat{V}(b_0))$

is not sufficiently small. Therefore, the approximation in these belief points needs to be improved.

The initial $\underline{V}$ corresponds to the value of a uniform strategy of the first player, and the initial $\bar{V}$ is a result of solving a perfect information refinement of the game.

3.3.1 Point-Based Update

A **point-based update** at belief point b can improve approximation at this point by updating $\underline{V}$ and $\bar{V}$ functions. After the update these functions needs to stay $(U-L)$-Lipschitz, which is required for proving the convergence of the algorithm.

The update of $\underline{V}$ adds an α-vector to the set Γ which corresponds to the value function of a Nash equilibrium strategy of the first player in $[H\underline{V}](b)$ (denoted $L\Gamma(b)$) computed from dual linear program (Equations 3.10, 3.11) mentioned before. This value function is linear and $(U-L)$-Lipschitz (Lemma 3.9), therefore, adding this α-vector to the set Γ preserves $(U-L)$-Lipschitz continuity of $\underline{V}$.

The update of $\bar{V}$ adds one point to the set Υ which corresponds to the evaluation of the value backup $[H\bar{V}](b)$ at belief point b. Denote this new point as $U\Upsilon(b)$. Computation of $[H\bar{V}](b)$ cannot be done directly by solving linear program (Equations 3.10, 3.11) mentioned before because $\bar{V}$ is not represented by α-vectors. The projection of beliefs to the lower envelope of $\bar{V}$ presented in [HB16] is used. Adding a point to Υ generally does not preserve $(U-L)$-Lipschitz continuity of $\bar{V}$ but it can be fixed by the following approximation:

$$\bar{V}(b) = \inf_{b' \in \Upsilon} \{\bar{V}(b') + (U-L) \cdot \|b - b'\|_2\} \tag{3.14}$$

The (Figure 2.3) shows the process of updating the lower and upper bound functions.

3.3.2 Forward Exploration

The HSVI algorithm needs to find the belief points which contribute to the insufficient approximation of the value function V^* in the initial belief.

Following **heuristic** provides a guideline for choosing such belief points.

The value backup operator H expresses the value at belief point b in terms of values of subsequent belief points $b^{a,o}_{\pi_2}$. When the value backup operator H is applied to the value functions $\hat{V}$ at belief point b, it also propagates the approximation error, therefore, the sufficient accuracy needs to be achieved also in beliefs encountered later.

The **forward exploration** simulates a play between the players where the second player follows a strategy obtained from the application of H on $\underline{V}$. If the approximation $\hat{V}(b)$ in belief b at time t (denoted as (b,t)) is not sufficient, it is said that it has positive **excess gap**:

Definition 3.13. Let ε be the desired precision and $R > 0$ be a neighborhood parameter. Let

$$\rho(t) = \varepsilon\gamma^{-t} - \sum_{i=1}^{t} 2R(U-L)\gamma^{-i} \tag{3.15}$$

The excess gap of in (b,t) is defined as

$$excess(b,t) = gap(\hat{V}(b)) - \rho(t) \tag{3.16}$$

The positive excess gap in (b,t) contributes to the insufficient approximation in the initial belief. In (b,t) the forward exploration chooses the subsequent belief (denoted as $(b^{a,o}_{\pi_2}, t+1)$) which has the highest positive observation/π_1-probability-weighted excess gap, i.e. the one which contributes to the insufficient approximation in the initial belief the most. If all subsequent beliefs have a negative excess gap, the forward exploration terminates because the point-based update will make the excess gap in (b,t) also negative.

3.3.3 Summary and Convergence of the HSVI Algorithm

The HSVI algorithm takes desired precision ε, neighborhood parameter R, and initial belief b_0 and returns an approximation of the optimal value function represented by upper and lower bound functions $\hat{V}$. The algorithm locally updates these bounds in specific beliefs, which are chosen by forward exploration heuristic. The forward exploration and local updates are

summarized in the following (Algorithm 5).

Algorithm 5: Explore Algorithm for POSGs: $explore(b, \varepsilon, R, t)$

Result: Updates the bound functions in specific beliefs which are found by forward exploration
Input : the root belief b, desired precision ε, neighborhood parameter R, depth t of belief b
Output : Updated sets Γ and Υ

$\pi_2 \leftarrow$ optimal strategy of the second player in $[H\underline{V}](b)$
$(a, o) \leftarrow$ according to forward exploration heuristic
if $excess(\hat{V}(b^{a,o}_{\pi_2}), t+1) > 0$ **then**
  $explore(b^{a,o}_{\pi_2}), \varepsilon, R, t+1)$
$\Gamma \leftarrow \Gamma \cup \{L\Gamma(b)\}$
$\Upsilon \leftarrow \Upsilon \cup \{U\Upsilon(b)\}$ and make $\bar{V}$ $(U-L)$-Lipschitz.

The complete HSVI algorithm for POSGs is described in the following (Algorithm 6).

Algorithm 6: HSVI Algorithm for OS-POSGs: $HSVI(G, \varepsilon, R)$

Result: Approximate value functions $\hat{V}$ such that $gap(\hat{V}(b_0)) \leq \varepsilon$
Input : Game $G = \langle \mathcal{S}, \mathcal{A}_1, \mathcal{A}_2, \mathcal{O}, \mathcal{T}, \mathcal{R}, \gamma, b_0 \rangle$, desired precision ε, neighborhood parameter R
Output : $\hat{V}$

Initialize the bounds $\hat{V}$
while $gap(\hat{V}(b_0)) > \varepsilon$ **do**
  $explore(b_0, \varepsilon, R, 0)$
return $\hat{V}$

The HSVI algorithm makes the excess gap negative in all reachable time beliefs, therefore, decreases the gap in the initial belief b_0. The following theorem provides theoretical results of the HSVI algorithm. Full theoretical discussion can be found in [HBP17].

Theorem 3.14 (Theorem 3 in [HBP17]). *HSVI algorithm for POSGs converges to the precision ε.*

Chapter 4

Approximation of the upper bound function of the HSVI Algorithm for OS-POSGs

The HSVI algorithm for OS-POSGs has insufficient scalability for games with a bigger set of states. The algorithm approximates the true value function V^* by a pair of convex functions. The lower-bound function is a PWLC function represented by a set of α-vectors and the upper-bound function is a lower convex envelope of a set of points. The updates of these functions present significant technical challenges.

This chapter discusses two possible modifications of the upper bound function: the first one rather basic, the other, using the Approximate Convex Hull algorithm.

4.1 Limitations and Basic Modifications of the HSVI Algorithm

The original algorithm periodically removes dominated vectors and points from their respective sets whenever their size grows by 20%. The removing of elements, of course, reduces the size of the sets, but also the size of the linear programs, therefore, the computation time. For the lower bound, the pruned vectors are the ones which are pointwise dominated by a single another vector. For the upper bound, the points $(b_i, \bar{v}_i)$ are dominated if and only if

$\bar{V}(b_i) < \bar{v}_i$. For any possible modification of the algorithm, it is necessary to preserve convexity of these upper and lower bound functions.

The first modification, which comes to mind, is an adjustment of the parameters of the pruning process. We can change how frequently is the pruning process called and also remove some additional elements selected at random apart from the dominated elements. Then we can observe how it affects the convergence of the algorithm. We will focus only on random deleting of the points from the upper-bound function because each vector from the lower-bound function represents the strategy of the first player; therefore, the constraint for the linear programs. Removing such vectors would require even more changes in the original algorithm, which creates an opportunity for further work.

Another way of modifying the upper-bound can be by removing the points which do not bring much value to the convex envelope of the upper-bound points. In computational geometry, the problem of finding a convex envelope of a finite set of points has been studied for a long time [BDH96]. The convex hull algorithms are usually quite effective only in spaces of lower dimension. Finding the convex envelope of a finite set of points in general dimension n is, on the other hand, a difficult task. General dimension algorithm is precisely what we need for constructing the upper-bound function. The reason for this is that the dimension corresponds to the number of states $|\mathcal{S}| + 1$, where the additional 1 is for the value of the upper-bound function in some belief b. Time complexity is another feature that we need to consider. The usual trade-off for a faster algorithm is accuracy. The next section describes an algorithm where the convex hull of a finite set of points is only approximated.

4.2 Approximate Convex Hull in High Dimensions

The **Approximate Convex Hull Algorithm in High Dimensions** [SV16] is an effective method for computing the approximate convex hull of the finite set of points in high dimensions with time complexity of $\mathcal{O}(K^{3/2}N^2 \log(\frac{K}{\varepsilon_0}))$, where N is a number of points and K is the number of iterations of the algorithm, which is usually significantly smaller than N.

The main task of the algorithm is to find a subset of points which defines the entire convex hull. These points are called **extreme**:

Definition 4.1. Let $X = \{x_1, ..., x_n\}$ be a finite set of points. The point x_i is called extreme point of X if it cannot be represented as a convex combination of points from the set $X \setminus x_i$.

The set of extreme points $\mathcal{E}$ represents the convex hull of X is a sense that $Conv(\mathcal{E}) = Conv(X)$, where $Conv(\cdot)$ denotes convex hull of a set of points. The basic operation needed in this algorithm is the euclidean distance of a point $x \in R^n$ to $Conv(X)$, which can be computed by a following quadratic program:

$$d(x, X)^2 = \min_{\alpha_i} \|x - \sum_{i=1}^{|X|} \alpha_i x_i\|^2 \quad s.t. \;\; \alpha_i \geq 0, \;\; \sum_{i=1}^{|X|} \alpha_i = 1 \tag{4.1}$$

Note that $x \in Conv(X)$ if and only if $d(x, X) = 0$.

Naturally, the key of the algorithm is to find smallest subset $\mathcal{E} \subseteq X$ such that $d(x, \mathcal{E}) = d(x, X)\ \forall x \in \mathbb{R}^n$. Such subset $\mathcal{E} \subseteq X$ would be accurate approximation if $d(x, \mathcal{E}) = 0\ \forall x \in Conv(X)$, but that is not needed for an approximate representation:

Definition 4.2. An ε-approximate convex hull of finite set X is the convex hull of minimal subset $\mathcal{E} \subseteq X$ such that $\forall x \in X$, $d(x, \mathcal{E}) \leq \varepsilon$.

4.2.1 Finding the Approximate Convex Hull

Denote $\mathcal{P}_C(X)$ as the set of all subsets of X with cardinality C or less. The basic approach for finding an ε-approximate convex hull of finite set X is to minimize ε for a fixed number of points, which can be formulated as the following optimization problem: find the subset $\mathcal{E} \in \mathcal{P}_C(X)$ which minimizes worst case distance of a point in X to the $Conv(\mathcal{E})$, formally:

$$\min_{\mathcal{E} \in \mathcal{P}_C(X)} \max_{x \in X} d(x, \mathcal{E}) \tag{4.2}$$

This is a combinatorial optimization problem which requires searching over each element of $\mathcal{P}_C(X)$. The **greedy version** will provide suboptimal solution but it will still achieve desired precision ε with a potentially larger cardinality than optimal. It finds each element of $\mathcal{E}$ sequentially, i.e. suppose that at step k we have found a set $\mathcal{E}_k$ with k elements. Then $\mathcal{E}_{k+1} = \mathcal{E} \cup \hat{x}$, where

$$\hat{x} = \arg \min_{x \in X \setminus \mathcal{E}_k} \max_{x' \in X \setminus \mathcal{E}_k} d(x', \mathcal{E}_k \cup x) \tag{4.3}$$

The greedy algorithm is initialized by $\mathcal{E}_0 = \emptyset$ and terminates either after fixed number of steps, or after reaching desired precision ε. Time complexity of the algorithm is $\mathcal{O}(CN^2)$ in order to reach cardinality of C. This is huge improvement. The following **improved greedy algorithm** reduces the search space even further.

Improved Greedy Algorithm

Let E be a matrix defined as follows:

$$E_{i,j} = d(x_i, \mathcal{E} \cup x_j)^2 \tag{4.4}$$

Then the index of the optimal point $\hat{x}$ defined in (Equation 4.3) can be obtained by:

$$\hat{j} = \arg\min_j \max_i E_{i,j} \tag{4.5}$$

In other words, the maximum value of each column of E is evaluated and then the minimum among those maximums is found. The matrix E has N^2 elements, i.e. N^2 evaluations of the distance function, but not all elements need to be found. Observe that the maximum of any subset of elements in a column provides a lower bound on the maximum over the entire column. The following method describes the entire process of finding the index $\hat{j}$:

1. Calculate the first row of E.
2. Find the minimum value of the first row and select the corresponding column as the potential candidate for $\hat{j}$
3. Compute the next element of that column and reevaluate which column has the smallest maximum. Set that column as the new potential candidate. If two columns have the same maximum, select one at random.
4. Continue until all elements of one of the columns have been found, and its maximum is less than or equal to the maximum of the other columns over the computed elements. In this case, $\hat{j}$ is the index of that column and the optimal value $\hat{\varepsilon}$ is the maximum value of that column.

Example 4.3. Consider a square matrix E with 16 elements in (Table 4.1). The first matrix shows all values that would normally need to be evaluated. Using the method just described, it is not necessary to evaluate all elements, as it is shown in the tables starting from left to right. In this case, $\min_j \max_i E_{i,j} = 2$ and $\hat{j} = \arg\min_j \max_i E_{i,j} = 4$.

The number of elements computed by this method ranges from $2N - 1$ in the best case to N^2 in the worst case. This method is formally described in

3	1	4	2
8	3	2	1
1	2	1	2
5	1	4	2

3	1	4	2

3	1	4	2
	3		

3	1	4	2
	3		1

3	1	4	2
	3		1
			2

3	1	4	2
	3		1
			2
			2

Table 4.1: Calculation of distance matrix $E_{i,j}$

the following (Algorithm 7).

Algorithm 7: Min Max of Matrix: $MinMaxOfMatrix(\mathcal{S}, \mathcal{E})$

Result: Finds the index $\hat{j}$ of the point from $\mathcal{S}$ to add as the next element of convex hull $\mathcal{E}$
Input : the set $\mathcal{S}$ of points from which to choose the new one, the set $\mathcal{E}$ of points of convex hull
Output : index $\hat{j}$ and min max distance $\hat{\varepsilon}$
Data: size of the matrix N, $\tilde{E}$ stores current maximum for each column, $\tilde{C}$ stores row indices of currently evaluated elements for each column

1 $N \leftarrow |\mathcal{S} \setminus \mathcal{E}|$
2 Calculate the first row of E
3 $\tilde{E} \leftarrow$ first row of E
4 $\tilde{C} \leftarrow \mathbf{1}^{1\times N}$
5 $\hat{j} \leftarrow$ index of minimum of $\tilde{E}$
6 $\tilde{C}_{\hat{j}} \leftarrow \tilde{C}_{\hat{j}} + 1$
7 **while** $\max \tilde{C} \leq N$ **do**
8 Calculate $E_{\tilde{C}_{\hat{j}},\hat{j}}$
9 $\tilde{E}_{\hat{j}} \leftarrow \max(\tilde{E}_{\hat{j}}, E_{\tilde{C}_{\hat{j}},\hat{j}})$
10 $\hat{j} \leftarrow$ index of minimum of $\tilde{E}$
11 $\tilde{C}_{\hat{j}} \leftarrow \tilde{C}_{\hat{j}} + 1$
12 $\hat{\varepsilon} = \tilde{E}_{\hat{j}}$
13 **return** $\hat{j}, \hat{\varepsilon}$

The algorithm can be further improved by finding and eliminating interior points.

Definition 4.4. A point $x \in X \backslash \mathcal{E}$ is an interior point of $Conv(\mathcal{E})$ if $d(x, \mathcal{E}) = 0$.

These interior points can be removed from the set X during the iteration without compromising the approximate convex hull. The interior points of $\mathcal{E}_{k+1}$ can be found via zeros in the $\hat{j}$-th column of the matrix E: Let $\mathcal{I}_k^*$ be a set of the interior points of $\mathcal{E}_k$ from previous steps. At k-th iteration the search over $(X \setminus \mathcal{I}_k^*) \setminus \mathcal{E}_k$ is performed and these interior points of $\mathcal{E}_k$, identified from zeros in the $\hat{j}$-th column, are placed in the set $\mathcal{I}_k$. Then $\mathcal{I}_{k+1}^* = \mathcal{I}_k^* \cup \mathcal{I}_k$ is defined and next iteration is performed. Also, some elements of from $\mathcal{E}_k$ may become interior points in future iterations. These points can be removed either at the end of each iteration or at the completion of the algorithm, which reduces the cardinality of the returned set of points.

The algorithm is initialized from an extreme point of $Conv(X)$, which can be obtained by the following theorem:

Theorem 4.5 (Theorem 1 in [SV16]). *Let X be a set of N points in $\mathbb{R}^n$. Any element of X, which has a minimum or maximum in one of its components, is an extreme point of $Conv(X)$.*

4.2.2 Summary of the algorithm

This algorithm computes the approximate convex hull for a given set of points, the desired precision and the maximum size of the convex hull. The

complete algorithm is formally described in the following (Algorithm 8).

Algorithm 8: *ApproximateConvexHull*$(\mathcal{S}, C, \varepsilon_{des})$

Result: Finds the approximate convex hull for a given set of points
Input : the set of points $\mathcal{S}$, the maximum size of the convex hull C, desired precision of the approximation ε_{des}
Output : Convex hull $\mathcal{E}$ of $\mathcal{S}$ and the error of approximation ε

```
Initialize E
S' ← S
while |E| < C and ε > ε_des do
    for x_i ∈ S' \ E and z_j ∈ S' \ E do
        Find ĵ and ε by Algorithm 7 where ĵ = arg min_j max_i E_{i,j}
        Using E_{i,ĵ} from previous step, find I_k so that if
          d(z_i, E ∪ x_ĵ) = 0 then z_i ∈ I_k
    S' ← S' \ I_k
    E ← E ∪ x_ĵ
    for p ∈ E do
        if d(p, E \ p) = 0 then
            E ← E \ p
    return E, ε
```

Example 4.6. Consider the following set of points $\mathcal{S} = \{A = [2, 3], B = [2, 0], C = [1, 1], D = [0, 1]\}$ visualized in the following (Figure 4.1).

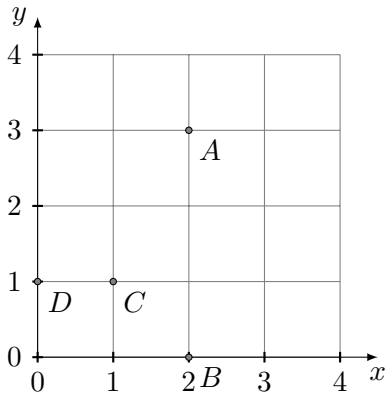


Figure 4.1: The points in the coordinate system

The task is to find the convex hull of $\mathcal{S}$ with $\varepsilon_{des} = 0$ and arbitrary cardinality. The algorithm is initialized by a point which has a minimum or maximum in one of its components, for example, A. The initial convex hull $\mathcal{E}_0 = \{A\}$. Now we need to find the second point using the matrix algorithm. The computation is presented in (Table 4.2). The elements of the matrices

are calculated by (Equation 4.4).

	B	C	D
B	0	2	4.5
C			
D			

	B	C	D
B	0	2	4.5
C	1		
D			

	B	C	D
B	0	2	4.5
C	1		
D	4		

	B	C	D
B	0	2	4.5
C	1	0	
D	4		

	B	C	D
B	0	2	4.5
C	1	0	
D	4	1	

Table 4.2: Calculation of the distance matrix for the 2nd point

From the last table, we can see that the second point to add to the convex hull is C and that the current min max distance ε, which is the maximum value of the column C, is 2. No points became interior in this iteration, therefore, $\mathcal{E}_1 = \{A, C\}$. The third point is found again by the matrix algorithm in (Table 4.3).

	B	D
B	0	2
D		

	B	D
B	0	2
D	1	

Table 4.3: Calculation of the distance matrix for the 3rd point

From the last table, we can see that the third point to add is B and that the min max distance $\varepsilon = 1$. No points became interior in this iteration, so the convex hull is $\mathcal{E}_2 = \{A, B, C\}$. The last iteration adds the remaining point D and so ε must be 0. Adding D to the convex hull makes C interior. Therefore, $\mathcal{E}_3 = \mathcal{E} = \{A, B, D\}$ and the algorithm is complete.

4.3 Using Approximate Convex Hull Algorithm in HSVI Algorithm for OS-POSGs

This section will describe how is the Approximate Convex Hull Algorithm from the previous section (Algorithm 8) used in the original HSVI Algorithm for OS-POSGs.

Every time the pruning of dominated points from the set Υ occurs, we call the Approximate Convex Hull Algorithm on the remaining set of non-

dominated points to decrease the cardinality of Υ even further. In this case, the Approximate Convex Hull Algorithm is not initialized by a single point but by a set of points that correspond to the pure beliefs. Such points will always be in the convex hull. The returned set of points forms an approximate convex hull of the original set of non-dominated points, and it is used as the new set Υ for the upper-bound function.

This modification gives us a total of three parameters to experiment with. The first one remains from the original algorithm: how frequently is the pruning part called. The other two are the parameters of the Approximate Convex Hull Algorithm: maximum size of the approximate convex hull and the desired precision of the approximation.

Chapter 5

Experiments and Evaluation

This chapter presents the experimental results of two modifications of the upper bound function from the previous chapter. We compare the modified algorithms to the original one. The first section is a brief introduction to the implementation of the approximations. Then the description of the games used in the experiments follows. The last two sections provide the experimental results of approximation by randomized point deletion and Convex Hull algorithm.

5.1 Implementation

The original HSVI algorithm is implemented in C++ language. The linear programs that occur in the algorithm are solved by IBM ILOG CPLEX. For the first approximation of the upper bound using randomized point deletion, we only modified the remove-dominated-points function to remove additional points. For the second approximation of the upper bound function, we implemented the Approximate Convex Hull algorithm also in C++ language. The quadratic programs for distance function are solved again by IBM ILOG CPLEX. The Approximate Convex Hull algorithm is then called by the remove-dominated-points function in the original algorithm. Note that the implementation part has not been deeply optimized since it was not the primary goal of this task.

5.2 Description of the Games

The experiments were performed on the series of games included in the original algorithm. These games are defined as follows.

Game 3

The Game 3 (peg3.posg) has 143 states, 21 partitions (the first player always knows the partition he is currently in. Partitions typically correspond to the perfectly observable components of the state description), 145 actions of the first player, 13 actions of the second player, 2 observations, 2671 transitions, 2671 rewards and discount factor 0.9500.

Game 4

The Game 4 (peg4.posg) has 363 states, 37 partitions, 290 actions of the first player, 18 actions of the second player, 2 observations, 8123 transitions, 8123 rewards and discount factor 0.9500.

Game 5

The Game 5 (peg5.posg) has 731 states, 57 partitions, 485 actions of the first player, 23 actions of the second player, 2 observations, 18335 transitions, 18335 rewards and discount factor 0.9500.

5.3 Experimental Results of the Approximation by Randomized Point Deletion

This section presents the experimental results of the approximation by randomized point deletion. The experiments were performed on the two following games. The goal of every presented setting of the algorithm is to achieve the precision $gap(\hat{V}(b_0)) \leq 0.5$.

5.3.1 Experiments on the Game 4

The experiments on the Game 4 were performed with 5 different following settings: while deleting dominated points, every i-th point will be deleted as well, where $i \in \{3, 5, 10, 25, 50\}$. The following table (Table 5.1) shows the results of the experiments compared to the original algorithm.

Algorithm	Convergence	Number of Iterations	Average Time of Iteration	Total Time	Precision	Number of Vectors in LB	Number of Points in UB
Original	YES	212	0,0974	25,3830	0,49653	3719	3529
Random 3	NO	2108	0,2988	634,5780	1,86705	21290	718
Random 5	YES	1586	0,3302	528,3660	0,49488	25234	1752
Random 10	YES	255	0,1082	32,4770	0,49852	4742	2776
Random 25	YES	217	0,1012	26,8540	0,49536	4023	3159
Random 50	YES	221	0,1001	26,9770	0,49666	3959	3379

Table 5.1: Experiments with approximation by randomized point deletion on the Game 4. In the Algorithm column, the Random i algorithm deletes every i-th point. The Convergence column presents weather the algorithm converged (precision ≤ 0.5) in the Total Time seconds. Average Time of Iteration is again in seconds. Precision column shows the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions.

We can see that in the case of Random 3, the algorithm did not converge in the presented time. The amount of deleted points was too high for it to converge. In the rest of the examples, the algorithm converged. The number of iterations is higher in the cases with more frequent deletion of the points. This makes sense because some essential points could be removed in the iteration and the approximation got worse. None of the examples converged faster than the original algorithm. In the cases where the number of removed points was high enough, the algorithm tried to compensate for the loss of the points by significantly increasing the number of vectors in the lower bound.

The following figures (Figure 5.1, 5.2) show dependence of the number of points and vectors on the iteration of the algorithms.

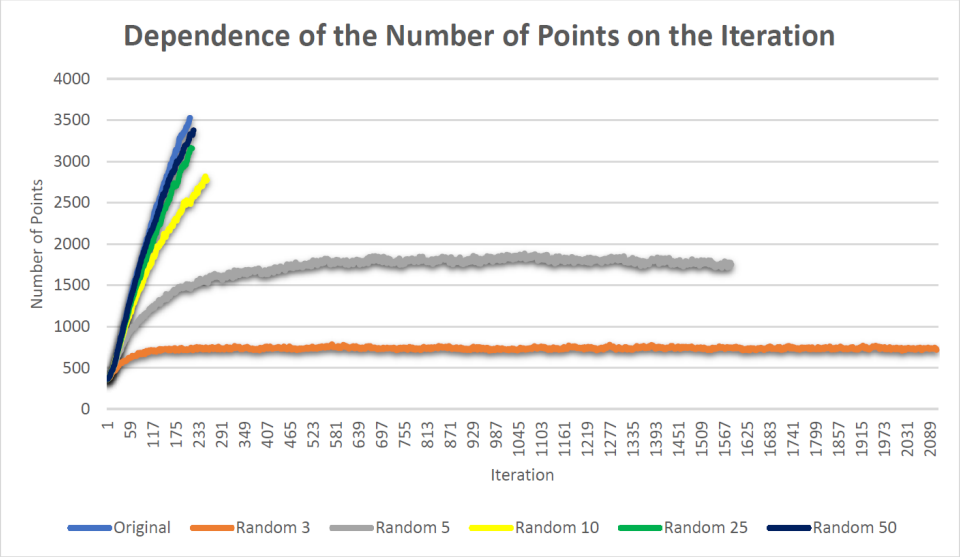


Figure 5.1: Dependence of the number of points on the iteration in the approximation by randomized point deletion in the Game 4

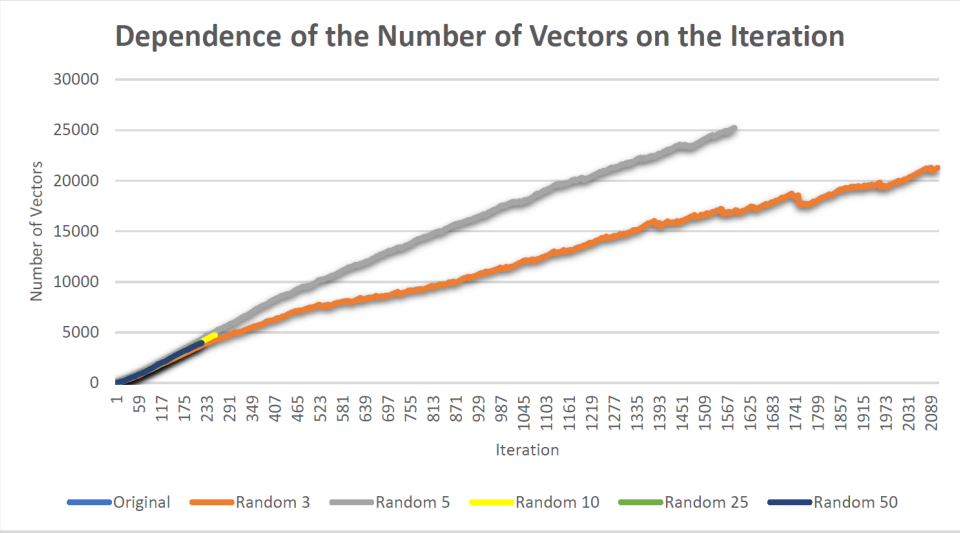


Figure 5.2: Dependence of the number of vectors on the iteration in the approximation by randomized point deletion in the Game 4

5.3.2 Experiments on the Game 5

The experiments on the Game 5 were performed with 5 different following settings: while deleting dominated points, every i-th point will be deleted as well, where $i \in \{5, 6, 10, 25, 40\}$. The following table (Table 5.2) shows the results of the experiments compared with the original algorithm.

Algorithm	Convergence	Number of Iterations	Average Time of Iteration	Total Time	Precision	Number of Vectors in LB	Number of Points in UB
Original	YES	573	0,2897	175,809	0,49894	13283	13052
Random 5	NO	2266	0,5917	1351,397	1,20504	48672	3529
Random 6	YES	1250	0,4267	543,0450	0,49844	29465	6277
Random 10	YES	650	0,3107	211,9350	0,49531	16409	9070
Random 25	YES	572	0,3331	202,0990	0,49675	14077	11229
Random 40	YES	578	0,3041	185,4940	0,49558	13700	12378

Table 5.2: Experiments with approximation by randomized point deletion on the Game 5. In the Algorithm column, the Random i algorithm deletes every i-th point. The Convergence column presents weather the algorithm converged (precision ≤ 0.5) in the Total Time seconds. Average Time of Iteration is again in seconds. Precision column shows the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions.

Similarly to the previous experiment, we can see that in the case of Random 5, the algorithm did not converge in the presented time. The amount of deleted points was too high for it to converge. In the rest of the examples, the algorithm converged. The number of iterations is higher in the cases with more frequent deletion of the points. None of the examples converged faster than the original algorithm. In the cases where the number of removed points was high enough, the algorithm tried to compensate for the loss of the points by significantly increasing the number of vectors in the lower bound.

The following figures (Figure 5.3, 5.4) show dependence of the number of points and vectors on the iteration of the algorithms.

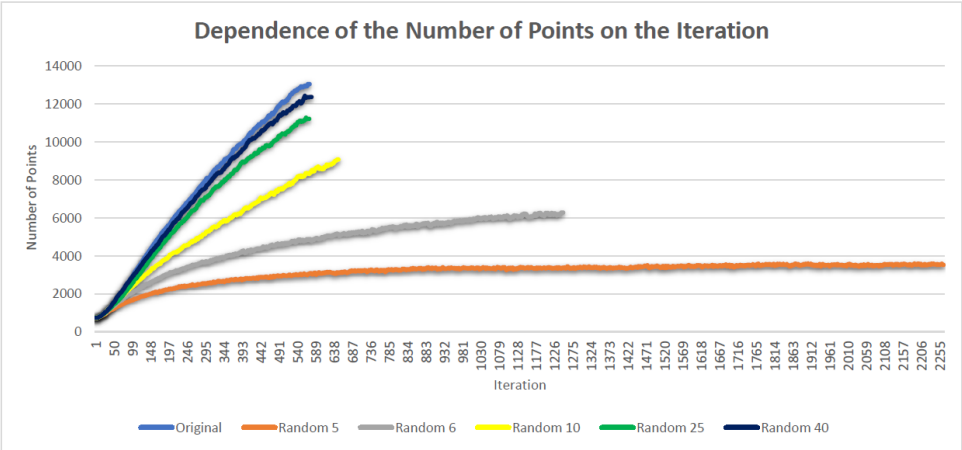


Figure 5.3: Dependence of the number of points on the iteration in the approximation by randomized point deletion in the Game 5

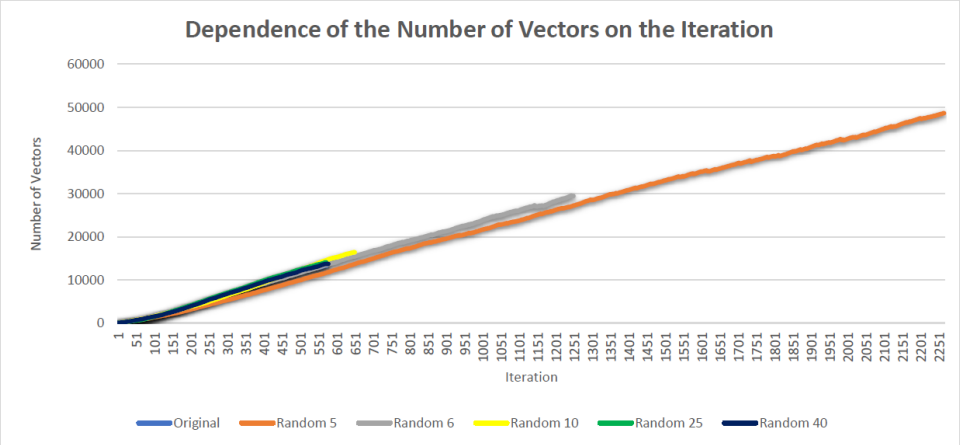


Figure 5.4: Dependence of the number of vectors on the iteration in the approximation by randomized point deletion on the Game 5

5.4 Experimental Results of the Approximation by Convex Hull Algorithm

This section presents the experimental results of the approximation by the Convex Hull algorithm. The experiments were performed on the four following instances. The goal of every presented setting of the algorithm is to achieve the precision $gap(\hat{V}(b_0)) \leq 0.5$.

5.4.1 Game 3 with Pruning 2 and Cardinality 50

In this case, the experiments were performed on the Game 3 with fixed pruning parameter 2 (the pruning is called when upper and lower bounds grow by a factor of 2) and maximal cardinality of the upper bound in every partition set to 50.

The experiments were performed with the precision of the Approximate Convex Hull algorithm $\varepsilon \in \{0.1, 0.2, 0.3, 0.5, 0.10, 0.20\}$. The following table (Table 5.3) shows the results of the experiments.

Algorithm	Number of Iterations	Average Time of One Iteration	Total Time	Time of the Original Part	Precision	Number of Vectors in LB	Number of Points in UB
Original	30	0,0215	2,33	2,33	0,49144	292	409
Approx 0,01	29	0,4144	13,22	2,28	0,48707	330	347
Approx 0,02	30	0,6735	21,18	2,44	0,48063	413	358
Approx 0,03	34	0,9749	20,76	2,68	0,45511	416	325
Approx 0,05	35	0,4956	18,52	2,62	0,45921	432	368
Approx 0,10	44	0,3381	16,32	3,15	0,47999	573	332
Approx 0,20	171	0,2985	52,57	11,19	0,3832	2129	318

Table 5.3: Experiments with approximation by Convex Hull algorithm on the Game 3 with pruning 2 and cardinality 50. In the Algorithm column, the Approx ε algorithm is called with the previously described settings: pruning 2, cardinality 50 and desired precision of the convex hull ε. Every algorithm converged to the precision ≤ 0.5 in the presented Total Time seconds. Average Time of One Iteration is in seconds. Time of the Original Part shows the fraction of the Total Time overlooking the time needed to compute the Approximate Convex Hull; the difference of these two columns is the time needed to compute the Approximate Convex Hull. Precision column presents the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions

The following figures (Figure 5.5, 5.6) show dependence of the number of points and vectors on the iteration of the algorithms.

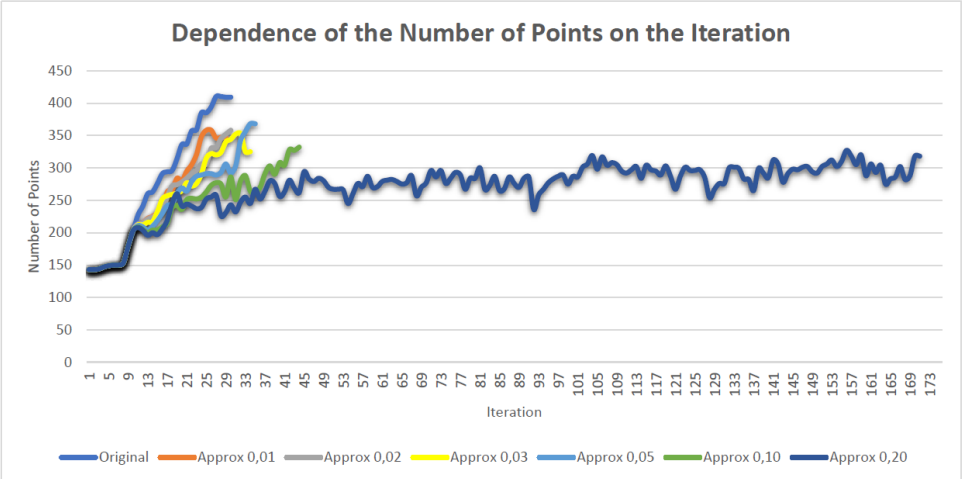


Figure 5.5: Dependence of the number of points on the iteration in the approximation by Convex Hull algorithm in the Game 3 with pruning 2 and cardinality 50

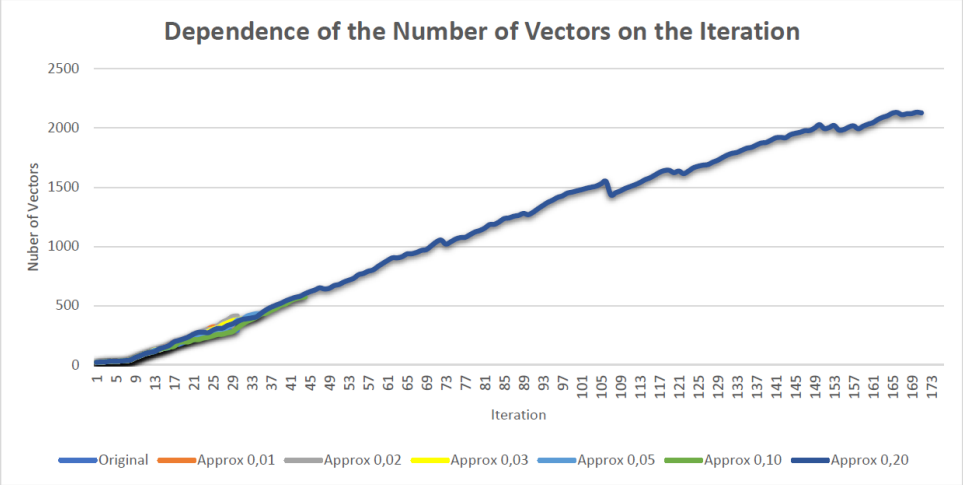


Figure 5.6: Dependence of the number of vectors on the iteration in the approximation by Convex Hull algorithm in the Game 3 with pruning 2 and cardinality 50

In every example, the algorithm converged in the presented time. The time needed to compute the Approximate Convex Hull causes the algorithm to be significantly slower than the original one. In every case, the number of points was decreased compared to the original algorithm, but the number of vectors increased. If we overlook the time needed to compute the convex hull, we can see that the time devoted to the original part of the algorithm is very similar to the original algorithm. In the case of Approx 0,01, we can see that this time is even slightly smaller. This can, of course, be caused by a statistical error.

5.4.2 Game 3 with Pruning 4 and Cardinality 50

In this case, the experiments were performed on the Game 3 with fixed pruning parameter 4 (the pruning is called when upper and lower bounds grow by a factor of 4) and maximal cardinality of the upper bound in every partition set to 50.

The experiments were performed with the precision of the Approximate Convex Hull algorithm $\varepsilon \in \{0.2, 0.4, 0.8, 0.15, 0.30, 0.60\}$. The following table (Table 5.4) shows the results of the experiments.

Algorithm	Number of Iterations	Average Time of One Iteration	Total Time	Time of the Original Part	Precision	Number of Vectors	Number of Points
Original	30	0,0215	2,33	2,33	0,49144	292	409
Approx 0,02	31	0,4062	13,82	2,26	0,49103	346	407
Approx 0,04	33	0,4295	15,55	2,56	0,42856	370	410
Approx 0,08	31	0,3445	12,25	2,65	0,45211	372	402
Approx 0,15	37	0,3013	12,67	2,81	0,42825	467	399
Approx 0,30	35	0,1655	7,47	2,75	0,47024	413	392
Approx 0,60	44	0,1433	7,96	3,31	0,43933	621	425

Table 5.4: Experiments with approximation by Convex Hull algorithm on the Game 3 with pruning 4 and cardinality 50. In the Algorithm column, the Approx ε algorithm is called with the previously described settings: pruning 4, cardinality 50 and desired precision of the convex hull ε. Every algorithm converged to the precision ≤ 0.5 in the presented Total Time seconds. Average Time of One Iteration is in seconds. Time of the Original Part shows the fraction of the Total Time overlooking the time needed to compute the Approximate Convex Hull; the difference of these two columns is the time needed to compute the Approximate Convex Hull. Precision column presents the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions

The following figures (Figure 5.7, 5.8) show dependence of the number of points and vectors on the iteration of the algorithms.

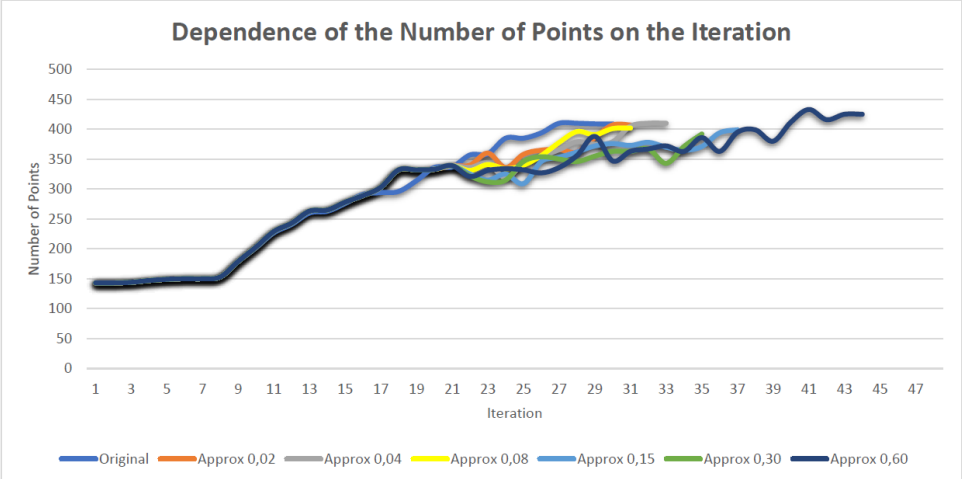


Figure 5.7: Dependence of the number of points on the iteration in the approximation by Convex Hull algorithm in the Game 3 with pruning 4 and cardinality 50

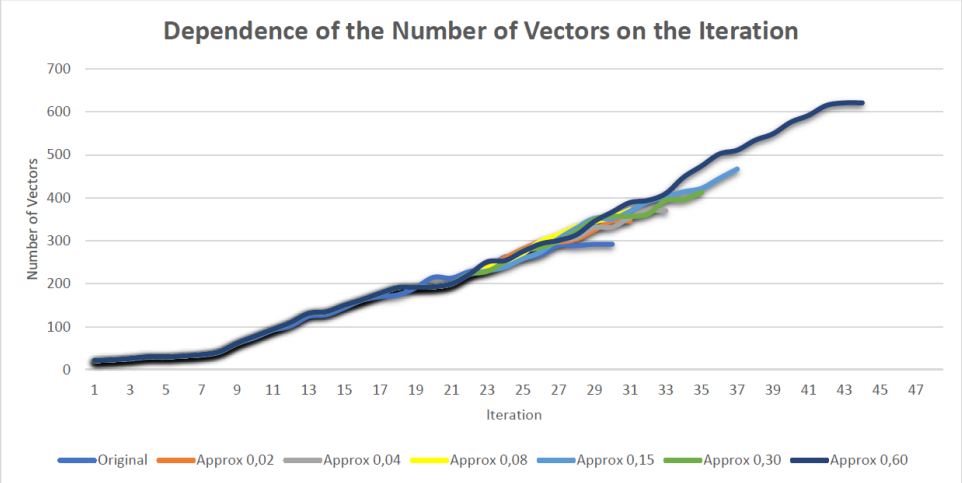


Figure 5.8: Dependence of the number of vectors on the iteration in the approximation by Convex Hull algorithm in the Game 3 with pruning 4 and cardinality 50

The results are very similar to the previous example. In every case, the algorithm converged in the presented time. The time needed to compute the Approximate Convex Hull causes the algorithm to be significantly slower than the original one. In every case, the number of points was very similar to the number of points in the original algorithm, but the number of vectors increased. If we overlook the time needed to compute the convex hull, we can see that the time devoted to the original part of the algorithm is very similar to the original algorithm. In the case of Approx 0,02, we can see that this time is even slightly smaller. This can, of course, be caused again by a statistical error.

5.4.3 Game 4 with Pruning 8 and Cardinality 100

In this case, the experiments were performed on the Game 4 with fixed pruning parameter 8 (the pruning is called when upper and lower bounds grow by a factor of 8) and maximal cardinality of the upper bound in every partition set to 100. We chose this setting to show the behavior of the algorithm depending on the desired precision of the Approximate Convex Hull algorithm because, in this game, the number of points in each partition rarely reaches 100.

The experiments were performed with the precision of the Approximate Convex Hull algorithm $\varepsilon \in \{0.2, 0.5, 0.8\}$. The following table (Table 5.5) shows the results of the experiments.

Algorithm	Number of Iterations	Average Time of One Iteration	Total Time	Time of the Original Part	Precision	Number of Vectors	Number of Points
Original	212	0,0974	25,38	25,38	0,49653	3719	3529
Approx 0,02	222	13,0304	2884,59	25,90	0,49524	4632	4230
Approx 0,05	239	12,1074	2886,44	28,54	0,48571	5052	3570
Approx 0,08	267	9,4436	2516,80	34,69	0,49077	6104	3289

Table 5.5: Experiments with approximation by Convex Hull algorithm on the Game 4 with pruning 8 and cardinality 100. In the Algorithm column, the Approx ε algorithm is called with the previously described settings: pruning 8, cardinality 100 and desired precision of the convex hull ε. Every algorithm converged to the precision ≤ 0.5 in the presented Total Time seconds. Average Time of One Iteration is in seconds. Time of the Original Part shows the fraction of the Total Time overlooking the time needed to compute the Approximate Convex Hull; the difference of these two columns is the time needed to compute the Approximate Convex Hull. Precision column presents the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions

The following figures (Figure 5.9, 5.10) show dependence of the number of points and vectors on the iteration of the algorithms.

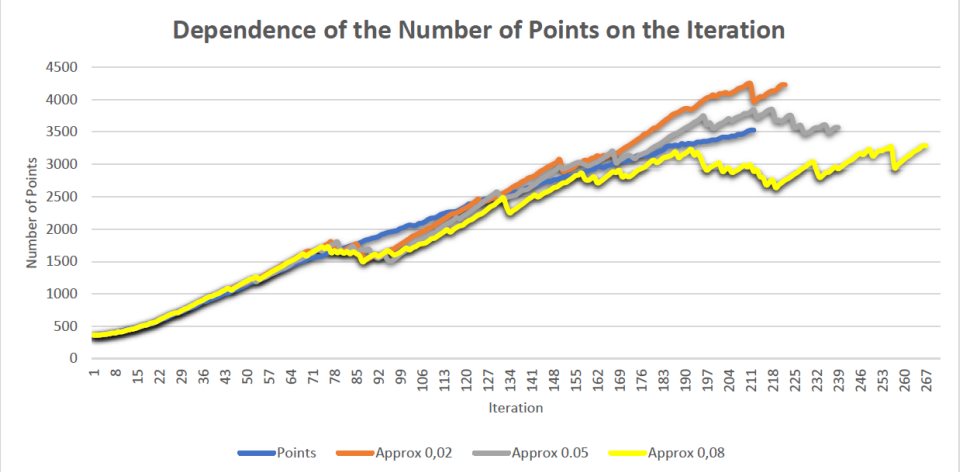


Figure 5.9: Dependence of the number of points on the iteration in the approximation by Convex Hull algorithm in the Game 4 with pruning 8 and cardinality 100

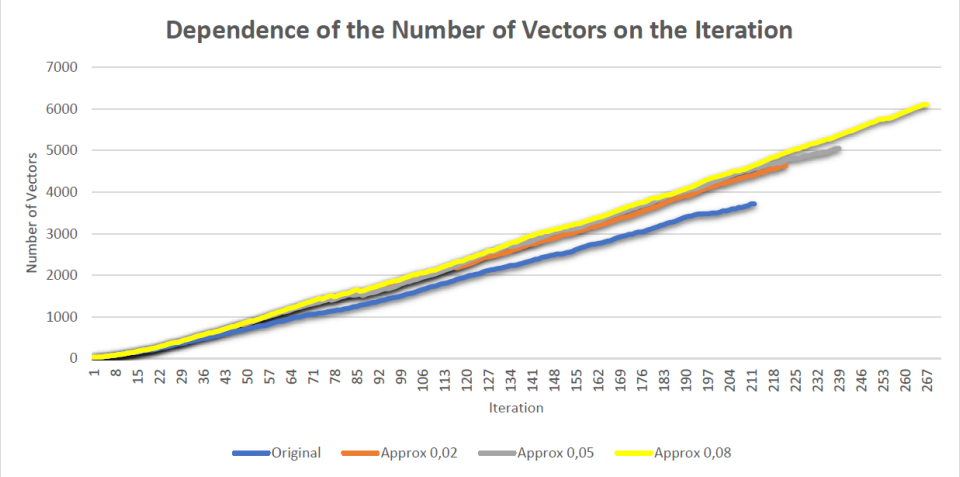


Figure 5.10: Dependence of the number of vectors on the iteration in the approximation by Convex Hull algorithm in the Game 4 with pruning 8 and cardinality 100

The results are very similar to the previous examples. In every case, the algorithm converged in the presented time. The number of points decreased only in the Approx 0.08 case. In all examples, the number of vectors increased. The number of iterations is roughly the same. Total time needed to compute the Approximate Convex Hull causes the algorithm to be significantly slower than the original.

5.4.4 Game 4 with Pruning 10 and Cardinality 100

In this last example, the experiments were performed on the Game 4 with fixed pruning parameter 10 (the pruning is called when upper and lower bounds grow by a factor of 10) and maximal cardinality of the upper bound in every partition set again to 100.

The experiments were performed with the precision of the Approximate Convex Hull algorithm $\varepsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ The following table (Table 5.6) shows the results of the experiments.

Algorithm	Number of Iterations	Average Time of One Iteration	Total Time	Time of the Original Part	Precision	Number of Vectors in LB	Number of Points in UB
Original	212	0,0974	25,38	25,38	0,49653	3719	3529
Approx 0,01	220	8,8211	1937,15	27,17	0,49092	4532	4638
Approx 0,02	222	8,3002	1839,14	25,01	0,49819	4256	4131
Approx 0,03	242	7,4079	1790,32	29,07	0,49167	4863	4529
Approx 0,04	224	6,4484	1442,85	26,46	0,49928	4549	4176
Approx 0,05	242	6,3036	1517,62	28,48	0,49759	4925	4214

Table 5.6: Experiments with approximation by Convex Hull algorithm on the Game 4 with pruning 10 and cardinality 100. In the Algorithm column, the Approx ε algorithm is called with the previously described settings: pruning 10, cardinality 100 and desired precision of the convex hull ε. Every algorithm converged to the precision ≤ 0.5 in the presented Total Time seconds. Average Time of One Iteration is again in seconds. Time of the Original Part shows the fraction of the Total Time overlooking the time needed to compute the Approximate Convex Hull; the difference of these two columns is the time needed to compute the Approximate Convex Hull. Precision column presents the achieved precision of the algorithm in Total Time. The last two columns present the number of elements of the lower bound (LB) and upper bound (UB) functions

The following figures (Figure 5.11, 5.12) show dependence of the number of points and vectors on the iteration of the algorithms.

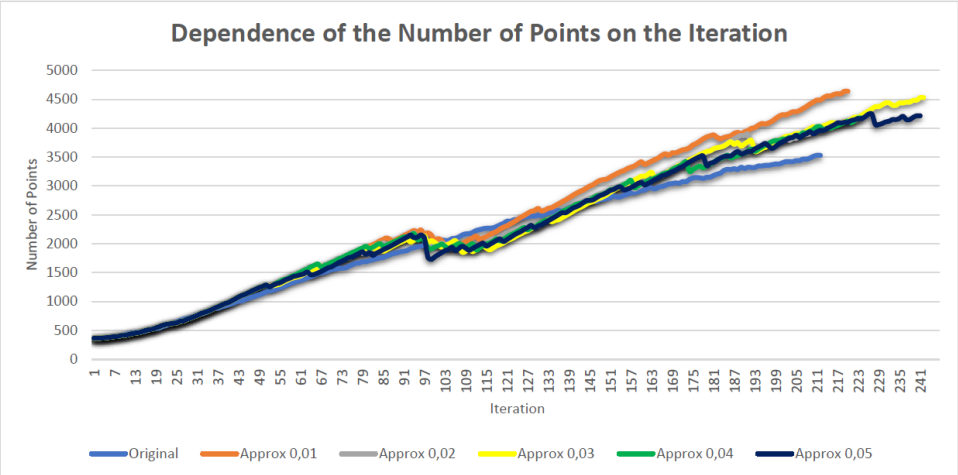


Figure 5.11: Dependence of the number of points on the iteration in the approximation by Convex Hull algorithm in the Game 4 with pruning 10 and cardinality 100

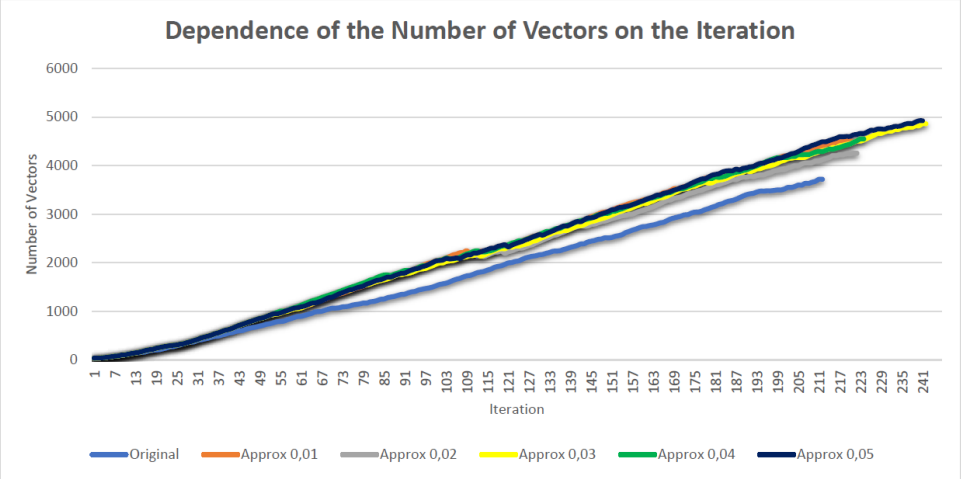


Figure 5.12: Dependence of the number of vectors on the iteration in the approximation by Convex Hull algorithm in the Game 4 with pruning 10 and cardinality 100

As we can see, the number of points and vectors increased in every case compared to the original algorithm. The number of iterations is roughly the same. Total time needed to compute the Approximate Convex Hull causes the algorithm to be significantly slower than the original. In the Approx 0.02 case, the time devoted to the original part of the algorithm was slightly smaller than in the original algorithm, but this can be caused by the statistical error.

Chapter 6

Conclusion

The Heuristic Search Value Iteration algorithm for POSGs approximates the true value function with upper and lower bound functions represented by points and vectors, respectively. These functions are updated by adding new elements to their sets, which is the key operation of the whole HSVI algorithm. Unfortunately, these updates present a bottleneck in the performance of the algorithm. We analyzed two possible approximations of the upper bound function represented by a set of points.

The first approximation focused on the randomized deletion of the points. The experimental results of this approximation are presented in Section 5.3. This approximation did not reduce the total computation time compared to the original algorithm but provided some interesting results. When the amount of deleted points were high enough, the algorithm tried to compensate for the loss of the points by significantly increasing the number of vectors in the lower bound. From this observation, we can deduce that further work should focus on the approximation of both bounds simultaneously.

The second approximation focused on the approximation by Convex Hull algorithm. The experimental results of this approximation are presented in Section 5.4. This approximation did not reduce total computation time as well; on the contrary, it significantly increased the computation time. The main reason for this is that at this moment, the methods for solving quadratic programs presented in the Approximate Convex Hull algorithm are not very effective. If we overlook the time that is needed for the solution of these quadratic programs, we got some interesting results. In three cases (Table 5.3 - Approx 0.01, Table 5.4 - Approx 0.02 and Table 5.6 - Approx 0.02) we got

slightly smaller computational time on the original part of the program. This is most likely caused by the statistical error. This result suggests that this approximation might be useful in the future when new methods for solving quadratic programs are possibly developed or for future research, which could focus on replacing the quadratic programs by different methods.

The overall results of the upper bound approximation are rather negative. The approximation of the upper bound is complicated. We can conclude that no matter how we approximate the upper bound function, we cannot get better results unless we also focus on the approximation of the lower bound function.

Appendix A

Bibliography

[BDH96] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, *The quickhull algorithm for convex hulls*, ACM Transactions on Mathematical Software **22** (1996), no. 4, 469–483.

[HB16] K. Horák and B. Bošanský, *A Point-Based Approximate Algorithm for One Sided Partially Observable Pursuit-Evasion Games*, Proceedings of the Conference on Decision and Game Theory for Security (2016), 435–454.

[HBP17] K. Horák, B. Bošanský, and M. Pěchouček, *Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games*, AAAI (2017), 558–564.

[K.C07] K.Ciesielski, *On Stefan Banach and some of his results*, Banach Journal of Mathematical Analysis 1(1) (2007), 1–10.

[KJT+09] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe, *Computing optimal randomized resource allocations for massive security games*, Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2009), 689–696.

[PJM+08] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus, *Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport*, Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2008), 125–132.

[RN09] S. J. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, 3 ed., Pearson, December 2009.

[SLB08] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, Cambridge University Press, New York, NY, USA, 2008.

[SS04a] T. Smith and R. Simmons, *Heuristic search value iteration for POMDPs*, Proceedings of the 20th conference on Uncertainty in artificial intelligence (2004), 520–527.

[SS04b] ———, *Heuristic search value iteration for POMDPs: Detailed theory and results*, Technical report, Robotics Institute, Carnegie Mellon University (2004).

[SV16] H. Sartipizadeh and T. L. Vincent, *Computing the Approximate Convex Hull in High Dimensions*, arXiv e-prints (2016), arXiv:1603.04422.

[VATS14] Y. Vorobeychik, B. An, M. Tambe, and S. P. Singh, *Computing Solutions in Infinite-Horizon Discounted Adversarial Patrolling Games*, Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS) (2014), 314–322.