

České vysoké učení technické v Praze  
Fakulta elektrotechnická  
Katedra kybernetiky



Bakalářská práce

**Prohledávání stromu informačních množin pro hraní  
obecných her s neúplnou informací**

**Information Set Search for General Game Playing with  
Imperfect Information**

Petr Švec

Vedoucí práce: Mgr. Viliam Lisý, M.Sc.

Studijní program: Otevřená informatika, bakalářský

Obor: Informatika a počítačové vědy

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

**Student:** Petr Švec

**Studijní program:** Otevřená informatika (bakalářský)

**Obor:** Informatika a počítačové vědy

**Název tématu:** Prohledávání stromu informačních množin pro hraní obecných her s neúplnou informací

### Pokyny pro vypracování:

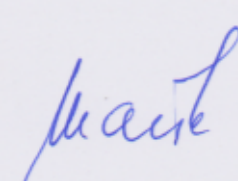
1. Student se seznámí s algoritmem „Information set search“ (ISS), používaným pro hraní specifických her s neúplnou informací. Například hry Kriegspiel (tj. Šachy s neúplnou informací).
2. Nastuduje hlavní principy, pravidla a požadavky soutěže v automatickém hraní obecných her (General Game Playing Competition), organizované při konferenci AAAI, s důrazem na hry s neúplnou informací o aktuálním stavu světa.
3. Navrhne vhodnou reprezentaci informačních množin pro obecné hry zadané v jazyku uvedené soutěže.
4. S použitím stávajících knihoven pro tvorbu hráčů naprogramuje obecného hráče používajícího na své rozhodování ISS.
5. Implementovaného hráče porovná s jinými existujícími hráči pro hraní obecných her s neúplnou informací.

### Seznam odborné literatury:

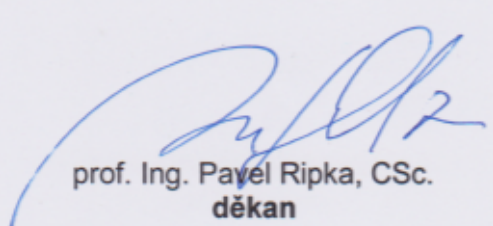
- [1] Genesereth, M., Love, N., & Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2), 62.  
<http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1813>
- [2] Thielscher, M. (2010). A General Game Description Language for Incomplete Information Games. AAAI 2010. <http://cgi.cse.unsw.edu.au/~mit/Papers/AAAI10a.pdf>
- [3] Parker, A., Nau, D., & Subrahmanian, V. (2010). Paranoia versus Overconfidence in Imperfect-Information Games. *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl* (pp. 63-84). Retrieved from <http://www.cs.umd.edu/~nau/papers/parker10paranoia.pdf>

**Vedoucí bakalářské práce:** Mgr. Viliam Lisý, MSc.

**Platnost zadání:** do konce zimního semestru 2012/2013

  
prof. Ing. Vladimír Mařík, DrSc.  
vedoucí katedry



  
prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 9. 1. 2012

## BACHELOR PROJECT ASSIGNMENT

**Student:** Petr Švec

**Study programme:** Open Informatics

**Specialisation:** Computer and Information Science

**Title of Bachelor Project:** Information Set Search for General Game Playing with Imperfect Information

### Guidelines:

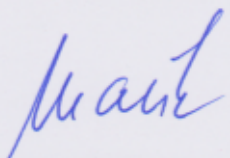
1. The student will learn the Information set search (ISS) algorithm, which has been successfully used to play several specific games, for example, Kriegspiel (i.e., imperfect information variant of Chess).
2. He will learn the main principles, rules and requirements of the General Game Playing competition organized at the AAAI conference. He will focus on imperfect information games.
3. The student will design a suitable representation of the information sets for games defined in the description language of the competition.
4. Using the existing libraries for creating players for the competition, he will implement general game player based on ISS.
5. He will compare the implemented player to other existing players for general game playing with imperfect information.

### Bibliography/Sources:

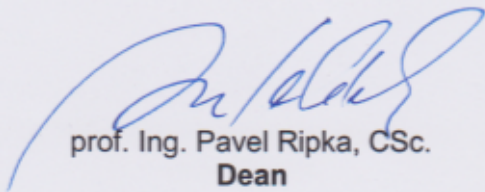
- [1] Genesereth, M., Love, N., & Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2), 62.  
<http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1813>
- [2] Thielscher, M. (2010). A General Game Description Language for Incomplete Information Games. AAAI 2010. <http://cgi.cse.unsw.edu.au/~mit/Papers/AAAI10a.pdf>
- [3] Parker, A., Nau, D., & Subrahmanian, V. (2010). Paranoia versus Overconfidence in Imperfect-Information Games. *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl* (pp. 63-84). Retrieved from <http://www.cs.umd.edu/~nau/papers/parker10paranoia.pdf>

**Bachelor Project Supervisor:** Mgr. Viliam Lisý, MSc.

**Valid until:** the end of the winter semester of academic year 2012/2013

  
prof. Ing. Vladimír Mařík, DrSc.  
Head of Department



  
prof. Ing. Pavel Ripka, CSc.  
Dean

Prague, January 9, 2012

## Prohlášení autora práce

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 dne 22. 5. 2012

Petr Šme

Podpis autora práce

## Acknowledgement

I'd like to thank to my thesis's supervisor Mgr. Viliam Lisý, M.Sc. for his valuable comments, advices and patience.

Petr Švec



# Abstrakt

Cílem General Game Playing je narozdíl od doménově specifických hráčů napsat hráče jednoho, který bude řešit široké spektrum her nebo problémů běžného života. Přes všechny výhody, které takový hráč nabízí je GGP stále většinou neprozkoumanou oblastí Umělé inteligence.

S uvedením druhé verze Game Description Language byla přidána podpora pro hry s neúplnou informací což má za následek existenci nových a mnohem složitějších her a problémů, od těch, které jsou přímočaré či technického rázu po ty, kde se plně projevuje složitost her s neúplnou informací.

V této práci popíšeme implementaci našeho hráče určeného speciálně na hraní her s neúplnou informací, který používá algoritmus zvaný Information Set Search.

Výkonnost našeho hráče testujeme na dvou hrách, z nichž první jsou Piškvorky, ve kterých nevidíme protihráčův tah a druhou je jednoduchá kartová hra.

## Klíčová slova

Game Description Language, Hraní obecných her, Hry s neúplnou informací, Information Set Search, Monte Carlo Tree Search





# Abstract

Unlike domain specific problem solving the General Game Playing (GGP) aims to use or write only one player for wide variety of games or real life tasks. Despite its possible wide applicability it still represent mostly unexplored parts of Artificial Intelligence. With the introduction of the second version of Game Description Language the support for Imperfect Informaiton Games (IIG) was added which results in much more complicated games and tasks with challanges ranging from technical ones and straightforward to difficult following from very nature of IIG. In this work we show implementation of our Imperfect-information player using Information Set Search (ISS) algorithm. Performance of our player is shown on playing Tic-Tac-Toe without knowledge of opponent's moves.

## Keywords

Game Description Language, General Game Playing, Imperfect Information Game, Information Set Search, Monte Carlo Tree Search



# Contents

List of Tables

List of Figures

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Theoretical Background</b>	<b>3</b>
2.1	Latent Tic-Tac-Toe . . . . .	3
2.2	Games . . . . .	3
2.3	Solution Concepts . . . . .	5
<b>3</b>	<b>General Game Playing</b>	<b>7</b>
3.1	General Game Playing . . . . .	7
3.1.1	GGP Competition . . . . .	7
3.1.2	Communication Protocol and Game Server . . . . .	8
3.1.3	GGP Player . . . . .	9
3.2	GDL . . . . .	10
3.2.1	GDL-I . . . . .	10
3.2.2	GDL-II . . . . .	11
3.3	GDL games . . . . .	11
<b>4</b>	<b>Information Set Search</b>	<b>13</b>
4.1	Minimax . . . . .	13
4.2	Monte Carlo Tree Search . . . . .	14
4.2.1	Discussion . . . . .	15
4.3	ISS . . . . .	16
4.4	MCTS-like ISS . . . . .	18
<b>5</b>	<b>Implementation</b>	<b>20</b>
5.1	Palamedes . . . . .	20
5.1.1	Architecture . . . . .	20
5.1.2	Reasoners . . . . .	20
5.2	Extension . . . . .	21
5.3	Managing big Information Sets . . . . .	22
5.4	ISS implementation . . . . .	23

<b>6 Experiments</b>	<b>25</b>
6.1 Methods of testing . . . . .	25
6.2 Comparison with TIIGR player . . . . .	26
6.2.1 Latent Tic-Tac-Toe . . . . .	26
6.2.2 Simple card game . . . . .	29
6.3 Tunning parameters . . . . .	30
<b>7 Conclusion</b>	<b>33</b>
7.1 Summary . . . . .	33
7.2 Future work . . . . .	34
<b>8 Bibliography</b>	<b>35</b>
<b>A List of Abbreviations and Acronyms</b>	<b>37</b>

# List of Tables

3.1	Server messages . . . . .	10
3.2	GDL-I terms . . . . .	11
6.1	Paranoid and Overconfident as XPLAYER . . . . .	28
6.2	Paranoid and Overconfident as OPLAYER . . . . .	29
6.3	Average utility in Simple card game . . . . .	30
6.4	Influence of Maximum simulations variable . . . . .	31
6.5	Dependency of Average utility on sample size with fixed number of simulations to 5. . . . .	32

# List of Figures

- 4.1 MCTS steps . . . . . 15
- 5.1 Palamedes components . . . . . 21
- 6.1 Comparison of Paranoid and Overconfident model with Motal’s player at ”X”  
position . . . . . 27
- 6.2 Comparison of Paranoid and Overconfident model with Motal’s player at ”O”  
position . . . . . 28
- 6.3 Comparison of Average Utility of Paranoid and Overconfident models on both  
positions with TIIGR player . . . . . 29
- 6.4 Influence of number of simulations on Average Utility . . . . . 31
- 6.5 Influence of Sample size on Average Utility . . . . . 32

*LIST OF FIGURES*

# Chapter 1

## Introduction

Men create various forms of games ever since we left trees. With the first computers it was logical to try to solve some of the games with their help. As the time passed the more and more complicated problems emerged and usually as soon as the problem was challenge no more the researchers moved to another even more complicated game.

In the last few years the advances in the field of Artificial Intelligence research allowed to play games which were considered to be unplayable without specialized hardware or algorithms before. Well known example is Deep Blue from IBM designed to play chess. Nowadays there isn't too many games in which the computer didn't defeated human players except for the ones such as Arimaa and similar ones which were created in the way to be difficult to play for computers and quite easy to humans.

The category of games in which the situation is bit different and which are gaining more attention of researchers are Imperfect Information Games (IIG). The difference compared to the classical games in which we know everything the game involves is the IIG provides only consequences of opponents' moves, changes and other interactions which occurs within the defined scope.

Unlike the players designed and implemented for specific task the General Game Playing (GGP) defines the opposite task. Instead of solving one concrete challenge by application of many various enhancements to the basic algorithm which work only in the specific problem or so called domain the GGP approach tries to create only one player which is able to solve wide variety of games and problems.

In several years after the introduction of GGP the support for the games of imperfect information was added. It leads to the increased complexity of tasks which can be solved but it also allowed the description of real-life tasks such as games based on Pursuit-evasion i.e. games in which there is typically many players in several teams which tries to achieve their goals by cooperation in partially observable environment.

Algorithms listed in this work are applicable or were designed for IIG playing and what is important they are general enough to be applicable in GGP. Despite, as we discuss further, it is still difficult to play most of these games. An example is Kriegspiel - an imperfect information variation of Chess or Bridge both being very difficult to play. Further in our work we mention techniques which allows at least to play them.



Except the algorithms and their implementation we discuss the realization of GGP match and we mention the conditions which the player must meet to be able to participate. Moreover we show how each definition of game looks like and which restrictions it must fulfil to be considered valid.

Two main contributions of our thesis are

- Implementation of the first Information Set Search based player in General Game Playing.
- Creating the extension to Palamedes framework which supports GDL-II.

The organization of the Chapters in the rest of this work is following. In the second Chapter we list several key concepts which we will use further. Chapter 3 is about GGP where we describe GGP Competition, technical aspect of communication with Game Server and both versions of Game Description Language (GDL). In Chapter 4 we list the game algorithms which we use in GGP player's implementation. Chapter 5 shows the implementation itself. First we describe Palamedes framework then the changes we made followed by discussion about the representation of Information Set (IS) and finally brief characterization of the implementation of MCTS-like ISS. Chapter 6 shows performance of our player compared to TIIGR in a card game and Tic-Tac-Toe where one doesn't see the opponent's moves. In the last Chapter - the Conclusion we summarize our achievements and list few possible subjects of our future work.

## Chapter 2

# Theoretical Background

In this chapter we list several key concepts necessary for further algorithms' description. We are going to describe chiefly them terms necessary for formal definition of Information Set Search i.e. Extensive Form Games and the related ones while concepts like Nash equilibrium are mentioned only briefly as they are not directly necessary.

### 2.1 Latent Tic-Tac-Toe

Despite it may seem quite unusual we start with description of game which we use further in this text in demonstrations of some situations and in experimentation.

This game is Latent Tic-Tac-Toe defined by Tomáš Motal [8]. Like in normal Tic-Tac-Toe there are two players, the X player and the O player. Same is also board size 3 x 3 and rules i.e. one need to place 3 of his/her move in row, column or diagonal to win. The difference is one player doesn't see turns of the other. This property of game often results in placing one's move on place where the opponent already did and thus there is necessary to have some kind of arbiter mostly for reasons described in section 3.1.2. Additional rule to this behaviour is the player which places his move on opponents takes turn again.

### 2.2 Games

The "game" term includes wide variety of problems ranging in difficulty from simple single player games to hard tasks simulating real world with multi-agent systems involving partial observability, nondeterminism and various forms of cooperation where most of these tasks are far beyond the scope of this work. We are going to focus especially on adversarial games where one's goal is to maximize his gain usually at the expense of other player(s). Before we start with formal definition we'll take a look at what a game is. Despite we won't define which properties (or relations to be accurate) such game must meet, we informally describe some basic attributes which are common in all games.

At the first place let's note the subject of our work are games with the sequential moves games i.e. those in which there is only one player who takes his turn at a moment however it doesn't mean we solve only some subset of possible games because without any loss on generality one can represent game with simultaneous moves as turn taking.

Restricting ourselves to this games means we can encounter both those where the players alternates in some (defined) order and those where player can make several moves consecutively. An example of the latter is Latent Tic-Tac-Toe presented in previous section where one plays again after he made move which was legal in the terms of game's rules but was not valid because opponent has already marked selected place before.

Now the games itself. Every one of them has following four attributes.

**Player** - In every game there is at least one player/agent. In common games humans usually take role of the player be it by their "physical presence" or by controlling some avatar via any user interface. However in games we are going to talk about these players are purely software.

**Action** - Each state has its related actions which can be made in it. It could be moves such as "turn left", "mark field with X" and so on.

**Payoff/Utility** - As soon as the match ends by application of sequences of players' actions each player receives some evaluation which represents his notion about this terminal state's usefulness. In all games which we will talk about every player knows the utilities of other players in every terminal node.

One could be tempted to add "state" item into this list however states are not common element in all definitions of games or even different types of games. On the other hand they exists in the games we are going to focus on and which are defined below.

While there is several options on how to define a game we selected the Extensive form which suits the needs of General Game Playing most. Unlike the Normal form the Extensive form is represented by game tree (or even by finite automaton eventually) where nodes represents states of the game plus overall progress in the game and the edges are the actions available at that nodes. Following formal definitions comes from [14]

**Definition 1** (Extensive form). *Finite imperfect-information extensive-form game is a tuple  $G = (N, A, H, Z, \chi, \rho, \sigma, \mathbf{u}, I)$ , where:*

- $N$  is a set of  $n$  players.
- $A$  is a set of actions.
- $H$  is a set of non-terminal choice nodes.
- $Z$  is a set of terminal nodes and  $H \cap Z = \emptyset$ .
- $\chi$  is a mapping  $\chi : H \rightarrow 2^A$ , which assigns to each node a subset of set of possible actions.
- $\rho$  is a mapping  $\rho : H \rightarrow N$ , which assigns to each node a player  $i \in N$  who chooses an action at that node.
- $\sigma$  is a mapping  $\sigma : H \times A \rightarrow H \cup Z$ , which assigns to each node and action a new node, both terminal and non-terminal. This assignment is unique i.e.  $\forall h, h' \in H$  and  $\forall a, a' \in A$ ;  $\sigma(h, a) = \sigma(h', a') \Rightarrow h = h'$  and  $a = a'$

- $\mathbf{u}$  is an utility vector  $\mathbf{u} = (u_1, \dots, u_n)$ , where  $u_i : Z \rightarrow \mathbb{R}$  and  $i \in N$
- $I = (I_1, \dots, I_n)$ , where  $I_i = (I_{i,1}, \dots, I_{i,k_i})$  is a set of equivalence classes on  $\{h \in H : \rho(h) = i\}$  with the property  $\chi(h) = \chi(h')$  and  $\rho(h) = \rho(h')$  whenever there exists a  $j$  for which  $h \in I_{i,j}$  and  $h' \in I_{j,i}$

As for the last point of the previous definition of the set  $I$ , one may also encounter definition using observations. We aren't going to use this one.

Notice the last point of the EFG definition says there are sets of nodes which represents our uncertainty i.e. says in which nodes we think we might be in. Such set is called *Information Set* (IS).

**Definition 2** (Zero-sum game). *Let  $G$  be the Extensive form game. Then we call it Zero-sum if and only if the  $\sum u_i = 0$  for every utility vector  $\mathbf{u}$ . All games which are not Zero-sum are called General-sum games.*

Next important term is the *perfect recall*. As its name suggests we call any game a *game of perfect recall* if all players in this game remember actions they made. Formal definition is as follows [14]

**Definition 3** (Perfect recall). *Let  $G$  be the imperfect-information game. Then player  $i$  has perfect recall if for any two nodes  $h, h'$  that are in same information set for player  $i$ , for any path  $h_0, a_0, h_1, a_1, \dots, h_m, a_m, h$  from the root of game to  $h$  and for any path  $h_0, a'_0, h'_1, a'_1, \dots, h'_{m'}, a'_{m'}, h'$  from the root of game to  $h'$  it must be the case that:*

1.  $m = m'$
2.  $\forall j \in \mathbb{N} : j \leq m$ , if  $\rho(h_j) = i$  (i.e.,  $h_j$  is a decision node of player  $i$ ), then  $h_j$  and  $h'_j$  are in the same equivalence class for  $i$
3.  $\forall j \in \mathbb{N} : j \leq m$ , if  $\rho(h_j) = i$  then  $a_j = a'_j$ .

$G$  is called a game of perfect recall if every player has perfect recall in it.

**Definition 4** (History). *History of node  $n$  is a sequence of actions leading from root node of game to  $n$ . According to the previous definition of game we write  $t = \langle a_1, a_2, \dots, a_k \rangle$  where  $a_i \in A$ .*

Let's note the history in a game of perfect recall is not same as history in imperfect recall game as it's unique in the former while there may exist several histories leading to same node in latter. Thus one can represent game of imperfect recall rather by some kind of graph or finite automaton instead of game tree whose nodes are usually understood to be unique.

## 2.3 Solution Concepts

Since all necessary concepts were defined in previous section we can finally begin with definition of solution concepts. As stated in [14] the term "solution concept" itself is kind of replacement for "optimal strategy". It comes from observation the more players are in

game the less our strategy influences game's state. Situation could also get even complicated when one encounters other difficulties such as partial observability and more.

Solution concept's cornerstone in Game theory is Nash equilibrium. Informally, this concept tells us which strategy(strategies) is most stable. Being in such equilibrium means none of players is willing to change his strategy without knowing the others would do the same. Good example of such situation is a monopoly position of few companies in market which have approximately equal share of clients. Then these companies don't need to change their strategy i.e. quality and amount of services provided for certain price since possible competition would decrease their profits.

Following two definitions are from [12].

**Definition 5** (Strategy). *Strategy of player  $i$  is a rule telling which action to choose at each instant of the game, given the Information Set.*

**Definition 6** (Strategy profile). *A strategy profile  $\sigma = (\sigma_1, \dots, \sigma_n)$  is an ordered set consisting of one strategy for each of the  $n$  players.*

Any further we'll use notation  $\sigma_{-i}$  for strategy profile without the strategy of player  $i$ .

Formal definition of Nash equilibrium requires *best response* to be defined first. Its definition is as follows [14]

**Definition 7** (Best response). *Player  $i$ 's best response to the strategy profile  $s_{-i}$  is a mixed strategy  $s_i^* \in S_i$ , such that  $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$  for all strategies  $s_i \in S_i$ .*

Now we can define Nash equilibrium.

**Definition 8** (Nash equilibrium). *A strategy profile  $s = (s_1, \dots, s_n)$  is a Nash equilibrium if, for all agents  $i$ ,  $s_i$  is best response to  $s_{-i}$ .*

## Chapter 3

# General Game Playing

This chapter introduces the General Game Playing and the Game Description Language. Section 3.1 is divided into several subsections in which we talk about GGP Competition, both game servers supporting this competition and communication protocol which each player must implement to be able to receive data from game server in next subsection and at last in general about requirements and restrictions laid upon any GGP player and we also mention few notable GGP players which won in last years.

Section 3.2.1 is about GDL where we discuss both language itself and properties which any game written in it must meet.

The last part - 3.2.2 is about extension of GDL-I, the GDL-II which contains new predicates for imperfect-information game support where we describe it by comparison with GDL-I.

### 3.1 General Game Playing

#### 3.1.1 GGP Competition

GGP as a field of research is quite old, dating to the late 60's [11]. Nevertheless the General Game Playing about which is this work relates to GGP Competition which was first introduced in [4].

Its originally intended purpose is to shift the "thinking" from system's/player's designers to software itself. This is achieved by sending game's description to player just at the beginning of the match. One may argue this is not absolutely true that everything is left for player to decide during the match as one may use domain independent heuristics. These are however not as much efficient as domain specific so the main problem remains.

First version of GGP Competition introduced in 2005 was only for Perfect Information Games. Since year 2011 this was extended to Imperfect Information Games. Because the imperfect-information extension is quite new there is only few players for the new version. Some of them both for PIG and IIG are mentioned in 3.1.3.

### 3.1.2 Communication Protocol and Game Server

Although the competition is organized by Association for the Advancement of Artificial Intelligence there isn't only one official Game Server nevertheless the communication protocol is common for all of them.

Game Server(GS) we are going to use and talk about is Dresden server so any further we talk about GS we have in mind just this one.

The role of server in General Game Playing is to provide some kind of interface which simplifies communication. If one would restrict himself/herself to 2-player turn-taking games the role of server is almost unimportant since the communication between players could be direct however it becomes important in games with simultaneous moves and multiplayer games. In the former there is no guarantee none of players is cheating i.e. sending his move after he receive opponent's and reacts to it in some way or gaining any other kind of intended or unintended advantage. On the other hand in multiplayer games(omitting multiplayer with simultaneous moves) the reason to employ server in communication is technical because without it the players would be forced negotiate order of players if it wouldn't be specified in rules and similar technicalities.

Probably the most important feature of server is processing responses from each player, verifying their correctness and sending proper information about state of game to opponents. In case the move the player sent was invalid in current state of game the server generates random one instead and again sends any observable informations both to opponents and to player who made invalid move.

Before we start with description of game messages we need to list names of variables we are going to use further.

**match\_id** is character string, number or combination of both which uniquely identifies current match.

**game\_description** variable stands for string description of selected game in Game Description Language which is discussed in section 3.2.

**start\_clock** is integer representing the time the player is given to perform any initialization procedures or game analysis. Its units are in seconds.

**play\_clock** is like start\_clock except it defines time given for one move.

**turn\_num** stands for number of move. Numbering goes from zero.

**my\_lastmove** is last move our player made. This is usefull especially in case we made illegal move or time ran out before we managed to send our move. In such case server makes random move as if it was ours.

**lastmoves** is list of all moves made in last turn. This variable is specific for Regualr GDL only. We mention it only for the sake of completeness.

**sees\_terms** variable is list of observations which resulted both from ours or opponents' moves.

Any GGP match is initiated by server which sends a start message. Start message is string containing all information necessary to properly initialize player. Its structure is as follows.

(START match\_id game\_description start\_clock play\_clock)

Every regular turn(nepresny - simultani) is started with play message. Unlike start message the structure of this one is dependent on whether we play perfect-information or imperfect-information game. Since our work is mainly about imperfect-information games we describe here only the structure for this case. Despite the version for PIG is listed in

(PLAY match\_id turn\_num my\_lastmove sees\_terms)

At the end of any match each player obtains stop message from server. Its structure is very same as the play's except there is STOP instead of PLAY. The purpose of stop message, except informing player about the end of the match, is to provide him also observation resulting from the last move in case it was opponent who took it so that he can determine utility vector since GS doesn't provide it directly.

From the technical aspect the communication itself is implemented as HTTP so that each player needs to embed at least the minimal web server. Sequence of messages contained in these HTTP requests which are sent between server and each player is following.

1. First comes start message with content mentioned above.
2. As soon as player parses message and initializes everything necessary he sends HTTP response with "READY" in its body.
3. When all players send their responses or start\_clock expires the server sends first play message with "NIL" in place of my\_lastmove and sees\_terms or lastmoves in case the game is in GDL-I.
4. Game proceeds with alternating play messages from the server and responses containing moves from players.
5. With game reaching one of its terminal states the server sends stop message to all players. The match ends.

### 3.1.3 GGP Player

There is few technical requirements and restriction laid upon GGP players. The domain independence mentioned in section 3.1.1 means not only difficulties with heuristics but during the process of reasoning too. Example of such case are games introduced in last years of GGP Competition whose GDL descriptions contained two disjunct games. Player's who didn't managed to find this out were forced to solve task with branching factor equal to the product of branching factors of each separate game. This problem is solved e.g. in [16] however we won't discuss it anymore since this work is about state space search algorithms not about reasoning.

There is several existing successful GGP players. However most of them is for perfect-information games only. Among these belongs following.



Message	GDL version	Structure
START	GDL & GDL-II	(START match_id game_description start_clock play_clock)
PLAY	GDL	(PLAY neco)
	GDL-II	(PLAY sees_terms)
	GDL-II new	(PLAY match_id turn_num my_lastmove sees_terms)
STOP	GDL	(STOP neco)
	GDL-II	(STOP sees_terms)
	GDL-II new	(STOP match_id turn_num my_lastmove sees_terms)

Table 3.1: Overview of possible messages from server.

- Flux Player - winner in 2006
- Cadia Player - won in 2007 and 2008
- Ary Player - winner in 2009 and 2010

The last two players are, one may say, typical players which uses same or similar algorithms like the one in following chapter [3], [9]. The Flux Player on the other hand is clearly logic-based, using Fluent Calculus [13].

In the IIG version of GGP there is very few players with Flux Player among them. Another representative of this category of players is TIIGR Player which uses reasoning like Cadia or Ary players but is able to determine set of possible states based on received observations. We'll briefly describe TIIGR further in Chapter 6 since we need few terms from the following chapters first.

## 3.2 GDL

### 3.2.1 GDL-I

GDL-I or simply GDL was introduced together with GGP Competition in year 2005. It supported only perfect-information games with several restriction we discuss in this section below. Despite these limitations the GDL is able to describe any usual perfect-information game like Tic-Tac-Toe, Chess etc., games with simultaneous moves which is Pacman for example and assuming we introduce additional player into game's description even the games with chance element.

Game Description Language consists of eight terms which are listed in table 3.2. First two are related to game initialization the following four specifies rules which holds in the course of game and the last two tells which states of game are terminal and which utility was achieved.

We describe these keywords only briefly as they aren't used any further.

The role terms specify the players in game. There is usually as many role terms as players. Only exception to this rule is case when we define additional player to simulate change nodes. Second term is init which describes initial state of the game which could be Pacman's position.

Term	Description
role(R)	R is a player
init(F)	F holds the initial position
true(F)	F holds in the current position
legal(R,M)	R can do move M in current position
does(R,M)	player R does move M
next(F)	F holds in the next position
terminal	the current position is terminal
goal(R,N)	R gets N points in the current position

Table 3.2: List of terms defined in GDL-I. Table comes from [15]

True, legal, does and next are very close. True provides information about statements which are true in current state and legal tells which moves are available in current state. Does describes changes to the state of world which are caused by application of any legal move and at last the next provides information about statements which holds in next state. Terms describing the end of game are terminal and goal where the former holds information about state of game in which it ends while the latter provides utility gained in the terminal state.

### 3.2.2 GDL-II

Few years later, in year 2011 GDL-II was introduced. It added two new predicates which allows description of IIG.

The first is sees(R,P) term which defines player's R perception P in the current state. Second is random which is term specifying random player for modelling of randomness.

With these new keywords one is able to define much more complicated game such as Battleship, Kriegspiel or Bridge.

## 3.3 GDL games

Each GDL game, no matter in which version was written must hold some rules. If it does then the game is called Well-formed. Here follows formal definitions as was introduced in [7].

**Definition 9** (Termination). *A game description in GDL terminates if all infinite sequences of legal moves from the initial state of the game reach a terminal state after a finite number of steps.*

**Definition 10** (Playability). *A game description in GDL is playable if and only if every role has at least one legal move in every non-terminal state reachable from the initial state.*

**Definition 11** (Monotonicity). *A game description in GDL is monotonic if and only if every role has exactly one goal value in every state reachable from the initial state, and goal values never decrease.*

**Definition 12** (Winnability). *A game description in GDL is strongly winnable if and only if, for some role, there is a sequence of individual moves of that role that leads to a terminal state of the game where that role's goal value is maximal. A game description in GDL is weakly winnable if and only if, for every role, there is a sequence of joint moves of all roles that leads to a terminal state where that role's goal value is maximal.*

**Definition 13** (Well-formed game). *A game description in GDL is well-formed if it terminates, is mono-tonic, and is both playable and weakly winnable.*

## Chapter 4

# Information Set Search

As the name indicates this Chapter is about Information Set Search (ISS) algorithm. However to correctly and formally describe it we need to remind Minimax and Monte Carlo Tree Search at first.

Minimax in section 4.1 is mentioned because ISS was introduced in minimax formula in section 4.3. In section 4.2.1 the MCTS algorithm is reminded which serves as an introduction to MCTS-like implementation of ISS in section 4.4.

### 4.1 Minimax

One of the simplest algorithms for adversarial search is Minimax. It's designed for zero-sum games and thus it assumes every player is maximizing its utility which also minimizes the opponent's. The basic Minimax is defined for 2-player games as following recursive formula

$$\text{min\_max}(h) = \begin{cases} \epsilon(h) & , \text{ if } S \in Z \\ \max_{a \in \chi(h)} \text{min\_max}(\sigma(h, a)) & , \rho(h) = \text{MAX} \\ \min_{a \in \chi(h)} \text{min\_max}(\sigma(h, a)) & , \rho(h) = \text{MIN} \end{cases}, \quad (4.1)$$

where  $\epsilon(S)$  stands for utility gained at terminal state  $S$ ,  $h \in H \cup Z$  and MAX and MIN are players alternating in specified order, where MAX is defined to be our player.

In case of n-player, general-sum game the situation is quite different. First the name "Minimax" is not accurate anymore since the game doesn't need to be zero-sum and thus the "minimizing" level doesn't have same effect like in Minimax and that's the reason why we prefer to call this algorithm *Backward Induction* like in [14].

The basic form of of formula is very close to Minimax

$$utility\_vector(h) = \begin{cases} \epsilon(h) & , \text{ if } h \in Z \\ \max_{a \in \chi(h)} utility\_vector(\sigma(h, a))_i & , \text{ where } \rho(h) = i \end{cases} \quad (4.2)$$

where  $i$  denotes  $i$ -th player.

Because we are in general-sum game only one value representing utility of node is not sufficient. This is solved by returning vector of values containing utilities for each player. In this formula the expression  $utility\_vector(\sigma(S, a))_i$  denotes the  $i$ -th component of utility vector which means each player is maximizing his utility.

Obviously such algorithm have quite narrow field of applicability since it assumes each player plays on his/her own. To include some cooperation into this algorithm one would need to alter maximalization formula and replace it with any similar e.g. the one maximalizing average utility of our and allied players.

Although there may be situation when both Minimax and Backward Induction are best options but this is not our case since as they searches a lot of unimportant nodes. Moreover in this form they doesn't count with depth-limited search for case of games with big state space. Nevertheless it fulfils our needs for ISS definition.

## 4.2 Monte Carlo Tree Search

Monte Carlo Tree Search belongs to the Monte Carlo methods class. Applied to game trees this technique involves running several random passes through this tree and then performs operations such as computation of average utility. When completed the children with highest average is selected. Used alone this method may easily get stuck in local maximum. Solution to this problem is Upper Confidence Bound applied to Trees algorithm [6] which is able to make good trade-off between exploration and exploitation and to avoid getting stuck in local maximum.

The UCT formula

$$uct\_val = v_i + C \cdot \sqrt{\frac{\ln N}{n_i}} \quad (4.3)$$

takes (usually average) utility of node  $i$ , here denoted as  $v_i$  and adds the UCT element where  $N$  is number of passes through parent node and  $n_i$  number of passes through this node. The constant  $C$  allows to control exploration-exploitation trade-off.

The MCTS is defined by the following four steps.

### 1. Selection

Selection always starts in the root node of current game tree. In this process we take the moves whose incident nodes are unexpanded first. As soon as all of them were visited the UCT formula is applied. The node with highest value is selected. This process repeats until the current node is leaf node of evaluated (sub)tree.

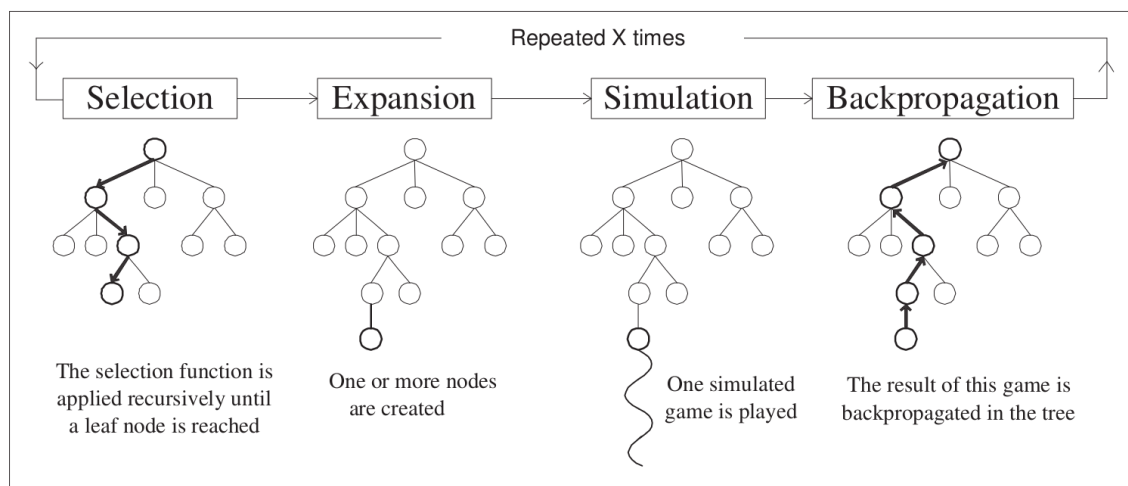


Figure 4.1: The four steps of MCTS. From [1]

## 2. Expansion

During the expansion we add node in which we ended during selection or all sibling nodes too. It's apparent in nodes near the current root there are no bigger differences between these two policies however the more deeper we get in game tree the lesser is the probability we will visit same node again. Thus in games which have deep tree it is inefficient to add all nodes at once as it consumes more memory without purpose.

## 3. Simulation

From the node or nodes we added in expansion steps the beams are run. These beams are random passings through game tree from selected node to terminal state. In this state we receive utility vector for all players and then we continue with backpropagation.

## 4. Backpropagation

In the last step - the backpropagation we take the utility vectors we received during the simulation. This vector is added to evaluations of all nodes we visited during the selection and node in expansion. In case we added more than one node during expansion only the respective vector is added.

These four steps runs may run either in loop conditioned by time or by the number of iterations depending on which task we solve.

Whenever we talk about MCTS any further we have in mind previous four steps with UCT algorithm used in the step of selection.

### 4.2.1 Discussion

Obviously the difference between MCTS and Minimax based algorithm is the MCTS is non-deterministic. Some of its advantages and disadvantages are the following.

### Advantages

- From the point of view of General Game Playing the most important aspect is domain independence and theoretical applicability to single player games. Next it is relatively simple extensibility to imperfect-information games which is discussed in section 4.4. Worth of note is also MCTS's ability to give relevant results even with small amount of iterations performed. This could be quite useful in GGP, in cases we play more complex game.
- Another advantage compared to Minimax is its property of non-uniform tree traversing. Thanks to UCT the MCTS chooses from the beginning the nodes which have higher expected utility which is noticeable especially in games with higher state space size such as Chess or Go where MCTS based players have success rate high enough to win even against their professional human opponents.

### Disadvantages

On the other hand there are some problems related to MCTS's non-determinism too.

- Among them belongs the speed of convergence in some games where a lot of iterations are necessary to obtain good results. Again this is especially problem in GGP.
- Next disadvantage, this time related to MCTS without additional domain information consists in games which have small amount of winning states compared to the size of entire state space. Typical representatives of these games are pacman, pursuit-evasion based tasks or if we decide to use MCTS for single player games then also passing maze with only one exit.

In the end we would like to mention the MCTS may have sometimes problems with numerical stability when there occurs periodical or non-periodical switching among two or more most prospecting results. Due to this problem the final selection of move we are going to play based on the results we achieved is usually implemented by choosing the move which was visited most often during MCTS.

## 4.3 ISS

Information Set Search is algorithm originally defined for 2-player, zero-sum games, which is based on Minimax [10]. Unlike the Minimax the ISS doesn't assume the opponent selects for us the worst-case action. Instead it uses opponent modelling to achieve optimal result. Before we start with description we need to define several concepts. All definitions listed in this section comes from [10].

We start with probability of history. Since the definition of ISS is only for 2-player games we slightly alter our definition of Extensive Form Game from section 2.2 and set  $N$  the number of players to 2. We'll denote first player A and second B. Thus  $\sigma_A$  is strategy of A and  $\sigma_B$  is strategy of B.

$$P(h|I, \sigma_A, \sigma_B) = \frac{P(t|\sigma_A, \sigma_B)}{\sum_{t' \in I} P(t'|\sigma_A, \sigma_B)}, \quad (4.4)$$

where  $P(t|\sigma_A, \sigma_B)$  is defined as

$$P(t|\sigma_A, \sigma_B) = \prod_{j=0}^{n-1} \sigma_{\rho(t_j)}(a_{j+1}|t_j) \quad (4.5)$$

In definition 4.5 the function  $\rho(t_j)$ , although in section 2.2 defined for nodes, returns the player who takes turn in Information Set in which the history  $t_j$  ended. Let's remind the history  $t_j$  is a sequence of actions applied to the root node and the respective nodes which resulted from these actions and index  $j$  denotes the  $j$ -th action in this sequence.

Now that we have defined the probability of history we continue with the Expected Utility(EU). But again, before we can define EU for the entire IS we need EU for single history which is as follows.

$$EU(t|\sigma_A, \sigma_B) = \sum_{a \in \chi(t)} \sigma_i(a|t) \cdot EU(\langle t, a \rangle | \sigma_A, \sigma_B) \quad (4.6)$$

Again we need to clarify some of the symbols.

As in equation 4.5 we made small "operator overloading" this times with  $\chi(t)$ . In EFG definition this function assigns moves to nodes but in this case it assigns moves to history. This is not quite as odd as it may seem since one may imagine history as an unique identifier of node in game of perfect-recall.

In this equation the strategy  $\sigma_i$  stands both for player A or B depending on which one of them takes turn in current node and finally symbol  $\langle t, a \rangle$  represents concatenation i.e. adding action to history  $t$ .

With definition 4.6 in mind the Expected utility for Information Set is

$$EU(I|\sigma_A, \sigma_B) = \sum_{t \in I} P(t|I, \sigma_A, \sigma_B) \cdot EU(t|\sigma_A, \sigma_B). \quad (4.7)$$

Equations 4.6 and 4.7 are obviously primarily for non-terminal nodes or IS. For the terminal ones the  $EU(t|\sigma_A, \sigma_B)$  is simply equal to the real utility  $U(t)$  or  $-U(t)$  in case  $\rho(t)$  is second player (we remind the game is zero-sum).

In most of the game the previous equations aren't much useful without some kind of evaluation function. Like in Minimax algorithm it's necessary to define evaluation function  $\epsilon(t)$  which will be used at defined depth  $d$ . These changes apparently involves only adding one more parameter representing depth both to 4.6 and 4.7 and one additional case in equation 4.6.

Now we with the previous equations we can define optimal set of moves.

$$\chi^*(I|\sigma_1, \sigma_2) = \arg \max_{m \in \chi(I)} EU(\langle I, a \rangle | \sigma_A, \sigma_B) \quad (4.8)$$



Applying equations 4.6 and 4.7 from terminal nodes allows to compute optimal strategy  $\sigma_A^*(a|I)$  assuming without any loss of generality we play as player A. Equation for its computation is following.

$$\sigma_A^*(a|I) = \begin{cases} \frac{1}{|\chi^*(I|\sigma_A^*, \sigma_B)|} & , \text{ if } m \in \chi^*(I|\sigma_A^*, \sigma_B) \\ 0 & , \text{ otherwise} \end{cases} \quad (4.9)$$

When all conditions are met this strategy is guaranteed to be optimal. Proof of this statement can be found in the work cited above.

Although the previous equation for optimal strategy may seem very good it requires knowledge of opponent's strategy or flawlessly precise opponent's model to be more accurate. There is of course very few occasions we have this kind of model so one need to rely on something more common. These "more common" things are two opponent models introduced in already mentioned work - the Paranoid and Overconfident.

**Paranoid model** is opponent model based on Minimax's property of paranoid assumption the opponent always makes the move which minimizes our utility. Despite its absolute dominance in perfect-information games in some imperfect-information game it achieves much worse than one might think. There are several reasons why it doesn't perform so well. First in PIG we can rely on our complete knowledge of opponent's intentions on the other hand in IIG we can only in limited degree. Each decision we make in IIG is based only on our assumptions and thus there is no guarantee the opponent has same knowledge (Information Sets) like we have. the "some IIG" we mentioned above are usually games with big Information Sets

**Overconfident model** may seem quite dumb since it is simple random selection pretending the amount of information is equal. However there are situations in which overconfident model outperforms paranoid. Among these belongs situations we have big information sets or when we iterate deeper in IS Tree the sets are getting bigger and provide less accurate information.

There are also some additional challenges like the size of IS. This and another are discussed later in Chapter 5.

## 4.4 MCTS-like ISS

Instead of Minimax algorithm as proposed in original work we can apply Monte Carlo Tree Search to Information Sets. With this change we traverse a tree composed of Information Sets and observations which alternates periodically. Each IS represents our state while observations and opponents' moves related to this observation represents our beliefs. Observation's selection strategy follows from opponents' model we have. Two basic, as mentioned before, are overconfident and paranoid. The first involves random selection as in Minimax ISS while the second is usually implemented as maximalization of negative UCT value.

Biggest difference compared to MCTS for Perfect Information Games is there are no opponents' nodes, only sees terms following from all possible combination of moves, both

ours and opponents'. According to these terms our new Information Set or Sets is/are created.

We summarize the properties mentioned above and describe MCTS applied to ISS in four parts like in section 4.2.1 more in depth.

### 1. Selection

First part of selection process is almost identical to regular MCTS - we compute UCT value for each move and select the one with highest.

What is different however is the action's/move's application doesn't result in new Information Set but in an Observation node. This Observation node incorporates all possible combinations of our player's observations combined with observations resulting from all possible opponents' moves. Let's note this is not same as a cross-product since our move, although legal based on the rules of game we play, doesn't need to be valid and the game's rules may allow us to make new move. This scenario results in sequence of two observations belonging to one player and thus observations can't be simply cross-producted.

After all observations are generated the selection strategy comes to play. As mentioned before two most basic strategies are overconfidence and paranoia. Based on its policy we obtain new Information Set.

### 2. Expansion

Expansion remains same like in regular MCTS except it works with IS instead of game nodes so we won't describe it in depth.

### 3. Simulation

Simulation slightly differs. Since one may have IS consisting of several game states there is an option to select only some of them and perform simulations in the same manner like in MCTS for Perfect Information Games. The number of selected states usually depends on played game.

### 4. Backpropagation

Even Backpropagation remains differing only in the data structure we update. So instead of game node we update values of IS or Observation node based on our implementation.

From these steps follows the only important change has been made in Selection step while others remained almost unaltered.

Although the IIG version of MC-UCT algorithm keeps its similarity to PIG version the behaviour is little bit different. While there was some reasonable trade-off between exploration and exploitation in games with Perfect Information in some Imperfect Information games this trade-off can be shifted towards the exploitation [2].

As for the size of expanded tree it is usually better to create smaller one because it is computationally more expensive to generate new IS from observations than running more simulation beams.

# Chapter 5

## Implementation

We start this chapter with brief description of Palamedes the framework designed specially for GGP in section 5.1. In next section 5.2 we write about the extension we made in Palamedes to work with IIG. The last section 5.4 is about implementation of Information Set Search using the Palamedes and extension we wrote. Part of this section is about tweaks whose goal is to improve the speed of algorithm.

### 5.1 Palamedes

#### 5.1.1 Architecture

Palamedes is Integrated Development Environment written by Ingo Keller [5] for perfect-information games which consists of three components. We are interested only in two of them the KIF Core and GDL Core. The third is UI plugin into Eclipse which is unimportant for us. Moreover from the two selected we'll discuss only GDL Core since it's the part we extended which is described in section 5.2.

Together with classes and methods for handling with GDL and representing states, moves etc. like in normal game it also provides methods for communication with Game Master/Server and parser to process incoming messages.

The most important Palamedes attribute is to provide additional level of abstraction between the resolver(s) and implementation of algorithm like MCTS or ISS. Basic components which one usually encounter during implementation of any algorithm both for PIG and IIG are listed in Figure 5.1.

#### 5.1.2 Reasoners

Although the components mentioned in previous subsection are important the major share of all work does reasoner also called resolver or prover.

Reasoner is usually some kind of library which implements algorithms for logical proving based on the resolution method.

There are three different resolvers bundled together with Palamedes and each of them has its advantages and disadvantages.

Interface	Content
IGAME	Facade for all game functionality
IREASONER	Interface for reasoner adapter implementations
IFLUENT	Fluent adapter interface, provided by a reasoner
IMOVE	Move adapter interface, provided by a reasoner
IGAMETREE	Interface for game tree implementations
IGAMENODE	Interface for a minimal game node functionality
IGAMESTATE	Interface for a minimal game state functionality
ISTATISTIC	General Interface for statistic support

Figure 5.1: The list of Palamedes Game Model interfaces [5]

**Prologprover** is the fastest of these three. Unlike the following this one is written, as its name suggests, in Prolog, namely TkEclipse Prolog, and thus the proper binaries and other components have to be installed especially the Java API which is necessary for communication between Palamedes and this Prolog.

Although it is most difficult to set correctly it is still worth the effort in case one plays any perfect-information game.

**Jocular** is the least interesting for us since it doesn't support the "+" character in Knowledge Interchange Format(KIF) syntax which makes it inapplicable for our purpose. As for speed this one is between Prolog prover and Javaprover

**Javaprover** is the last one. It's easiest to use as it doesn't require any other components to run. The version of Javaprover bundled in Palamedes doesn't support GDL-II however it can be easily extended since this all necessary classes are part of GGP Project and available at <http://sourceforge.net/apps/ggpserver>.

## 5.2 Extension

As we mentioned before the Palamedes supports only perfect-information games.

From section 5.1.2 follows the only reasoner we can use is Javaprover. All one need to do is copy source obtained from GGP Project server into prover's packages. Still this isn't enough as it's also needed to get the sees terms in proper format. Fortunately for us this task was done by Tomáš Motal in [8].

With this function we can now obtain observations following from applying any legal move in its related node and cluster the moves according to the observations.

The Palamedes extension we are going to briefly describe is a set of classes which extends the functionality of the existing ones, replaces them or adds new functionality.

First we extended the original implementation of IGame interface. It provides almost same operations like the original one but it performs them on IS not on game nodes. However

one can still call the methods of the original (super)class. The most important functionality which this class provides are the clustering functions which categorize the move or moves into groups according to the observations which follows from each move. We provide four different versions of this method. Two of them return an array of groups of new moves and the other do the same but returns directly new IS following from those moves.

We have also written class which is a wrapper for sets of all moves legal in any IS or belief set. This is not so crucial for our moves but in belief sets there may be lot of different moves.

Another new classes are those for Informaiton Sets and for Information Set Tree. While we use the former very often this can't be said about IS Tree. Since it can't extend the original (it is irrational) GameTree class as the IS Tree is much different. With this however we lose the caching which is implemented in GameTree class.

### 5.3 Managing big Information Sets

In the Introduction we mentioned two games - Kriegspiel and Brige and also that this games are very big. In this case the big means they have IS with sizes up to millions or billions of nodes. With such IS it is impossible to effectively play the game or even to play at all. For this reason the methods and techniques which allows to manage so big IS.

In the following list we discuss three options on how to solve the presented problem. First two are presented only as an possible option since it's unclear if they will work.

#### 1. Game decomposition

Game decomposition is technique presented in [16] which takes the game apart and returns the subgames. Each of these subgames can be then solved almost as a separate problem. When all the separated problems were solved or more realistically there were performed searches in each of these subgames which returned some result then so called global game search is performed. According to the statement of author this method of game solving is as optimal as one can get by running state space search algorithm on original game.

Since the Game decomposition was designed for GDL-I we can't be sure the optimality statement will hold in IIG too.

#### 2. State agregation

State agregation is technique used in situations when there is lot of similar states in the Information Set. Using it allows to perform any reasoning with the entire groups of nodes which are in the same agregated state. Its applicability in GGP is very questionable since there is no general algorithm as far as we know.

#### 3. Statistical sampling

Unlike the previous two the statistical sampling method is usable without changing the prover as it can be implemented directly in state space search algorithm.

Sampling can be made in two possible ways. First one is static selection when one selects subset of all nodes in IS. The second is iterative selection which iteratively takes random node from IS and adds it to the sampled set.

For the implementation we have chosen statistical sampling with the static selection. We'll describe the reason why we did so in the next section.

## 5.4 ISS implementation

Before starting with ISS we implemented common MCTS algorithm for PIG. As we showed in section 4.4 there is quite very few differences between MCTS-like ISS and MCTS itself. Using extensions we described in section 5.2 we altered respective sections in original MCTS algorithm.

The first and the most significant change was adding the function which generates new IS according to the observations which are generated both from our and opponents' moves. This function is provided current IS and a move which is legal in this set and uses them to perform something like breadth-first search except it handles specially the current IS as follows.

First it generates observations following from our move applied to the current set. From the generated observations originate new sets and from these sets in loop we generate new observations and new sets resulting from them until all observations are generated i.e. until we reached the terminal sets which are ours IS.

This function is applied whenever we are expanding IS or technically it may be even considered to be expansion itself.

Next part we changed is simulation. Since the IS may contain many nodes we can select from how many of them we would like to run the random passes. For the selection of nodes we uses the most basic method which is the generation of random numbers representing the indices of nodes with allowed repetitive selection of one node.

In MCTS-like ISS there is possible to set several parameters and some of them were already mentioned.

1. The C constant which remains from MCTS.
2. Sample size which was mentioned in previous section.
3. The number of random passes which we run from IS during the selection.
4. The maximum depth from which the algorithm performs the expansion no more.
5. The number of visits of unexpanded IS necessary for it to be allowed to get expanded.

From this list we selected only #2 and #3 and used them in the implementation.

The first one we want to discuss is statistical sampling and the parameter of the size of sample. As we mentioned in the previous section we have chosen statistical sampling with static selection over the iterative one. The reason why we did so is the iterative's unclear handling with belief sets and observations during the expansion and thus we use static. With the static version of statistical sampling one has another two options on how to apply the sampling on moves.

First option is random selection from all moves before they were clustered according to observations, preferably with unique selection i.e. without repeating. Second approach cluster moves first and then performs sampling on each group separately. In our implementation we use the first approach.

The second parameter we'll talk about is the maximum number of random passes through game tree to the terminal nodes which are run in simulation step.

The real number of performed passes is computed as minimum of the size of IS and the maximum simulations parameter and then the desired number of simulations is run. The nodes from which the simulation are run are selected randomly with allowed repetition. Obviously implemented this way the maximum number of random pass is dependent on the size of the sample.

# Chapter 6

## Experiments

As indicated subsection 5.4 of Chapter 5 there is quite many parameters to be set. In section 6.1 we list those which we will be going to test together with description of our testing methods. Next section - 6.2 is about the testing of the usually most significant variable which is time. And finally in the last part of this chapter we show how settings of the other parameters influences overall result.

### 6.1 Methods of testing

In Chapter 5, namely in section 5.4, we mentioned several parameters which can be changed to tune our player's performance. Despite we won't test all of them except the following.

1. Time variable
2. Number of simulations performed in each IS.
3. Number of node in each sampled IS.

The time variable is in most game playing algorithms the crucial factor and thus we discuss it in separate section.

Parameters of performed simulations and the size of sampled IS are discussed in section 6.3.

All tests were performed on PC with Core2 Duo T9600@2.8Ghz with 8GB RAM. Maximal Java Heap space was set to 5.5 GB while initial to 1GB.

Let's note the memory was used at max in cases of longer times per move. The reasons why it occurs are mentioned in 5.2.

Each test was done by script running 20 matches for selected setting.

The main part of experiments, involving all those parameters mentioned above, was done in Latent Tic-Tac-Toe described in section 2.1. Any results obtained from this game are 0, 50 or 100 where 0 is loss, 50 draw and 100 is win.

Part of the testing is also comparison with Motal's TIIGR player introduced in [8].

However this comparison isn't done directly i.e. by setting players to play against each other but by comparing their averaged results in matches against random player in both positions. The reason why we decided to do so was that because TIIGR was written with



old version of PLAY message we would need to replace old TIIGR's parser with our to process correctly messages from server.

To discuss the results from experiments we need to describe, at least briefly, TIIGR first. The TIIGR player is also implemented with Palamedes framework. It uses algorithm called Perfect information sampling. This algorithm works only with one IS which is the root node of the (imperfect-information) game tree. From this IS it selects several nodes according to its policy and from them it searches regular PIG tree. Each of this searches returns best found move.

The selection of the final move is done in following way. From each perfect-information search we take the number of passes made through the best move. This number represents the confidence. Finally the move which has the highest average of confidence of its values is selected.

Let's note the selection policy we mentioned can be either random selection which is called overconfident or selection which favours opponent which is called paranoid from the assumption the opponent makes his best move.

Last two things we would like to mention in this section is naming convention. Anytime we talk about XPLAYER or OPLAYER we have in mind player making "X" as his move or "O" respectively. And the second one: all experiments were made in matches with random player.

## 6.2 Comparison with TIIGR player

### 6.2.1 Latent Tic-Tac-Toe

In this section we would like to show not only the influence of time variable on average utility but we also want to compare our algorithm with the one called TIIGR player implemented in [8]. This comparison won't be absolutely correct since the time settings which we use starts at 5 seconds per move while TIIGR started with 100 milliseconds per move.

The reason behind the decision of setting the first time value so high is there are situations in which our algorithm isn't able to expand all moves and the possible observations leading to new IS from root IS in time lower than 5 seconds. Among them belong cases such as big initial Information Set and setting of parameters we use.

First of our tests is comparison of Paranoid and Overconfident opponent model with Motal's player in Figure 6.1. Before we start we would like to remark the only three measured results of player with overconfident opponent model were caused by our inability to perform the fourth set of matches since we ran out of memory.

In the mentioned chart one can observe quite slight differences between two opponent models which are cause by the advantages both of them have.

First the Paranoid model's advantage is its capability to perform much better in Information Sets with bigger amount of information i.e. those consisting of fewer nodes which is the case of XPLAYER starting the game and having the IS of size 1.

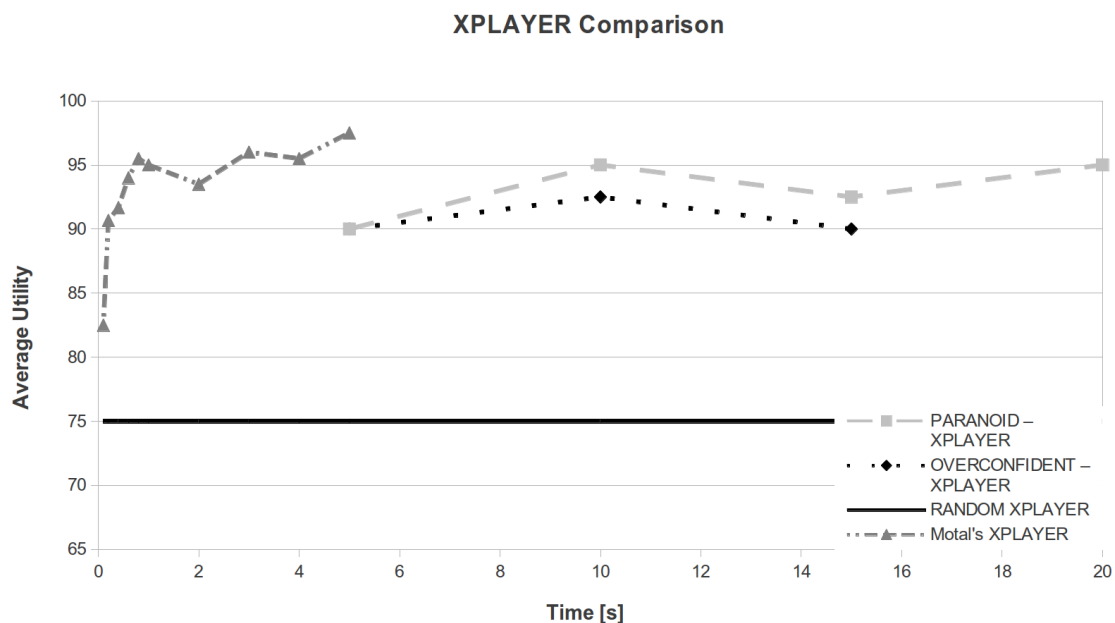


Figure 6.1: Comparison of Paranoid and Overconfident model with Motal’s player at ”X” position

On the other hand Overconfident has perfect opponent model since our algorithm is playing against random player. It means he should play optimally in theory. In practice or in our implementation to be more specific this is not absolutely true since we don’t use the original formula mentioned in section 4.3 but MCTS-like variant which implies we may get some better results after certain amount of iterations.

Informally we observed the version of our player which used Paranoid model placed ”X” into the center of board more often than the Overconfident. We think that this and the number of iterations which is our algorithm able to make caused the latter to perform slightly worse than the former.

Compared to the Motal’s player our implementation of ISS gives worse average results. This is very likely caused by ineffectivity of implementation of functions presented in section 5.2 since we generate lot of duplicit states due to non-present caching which decreases our algorithm’s overall performance.

In Figure 6.2 one can see the results of two presented opponent models from matches when playing as OPLAYER. This time the Overconfident gives much better results than the Paranoid. As was already told the former has advantage of having precise model of opponent which gives better results than the latter does. Playing at ”O” position i.e. playing as the second player isn’t for Paranoid model so much favourable as is playing at first position since it already begins in Information Set of size 9 as it didn’t see opponent’s move which means lesser precision of the paranoid assumption the opponent make such move which is the worst for us.

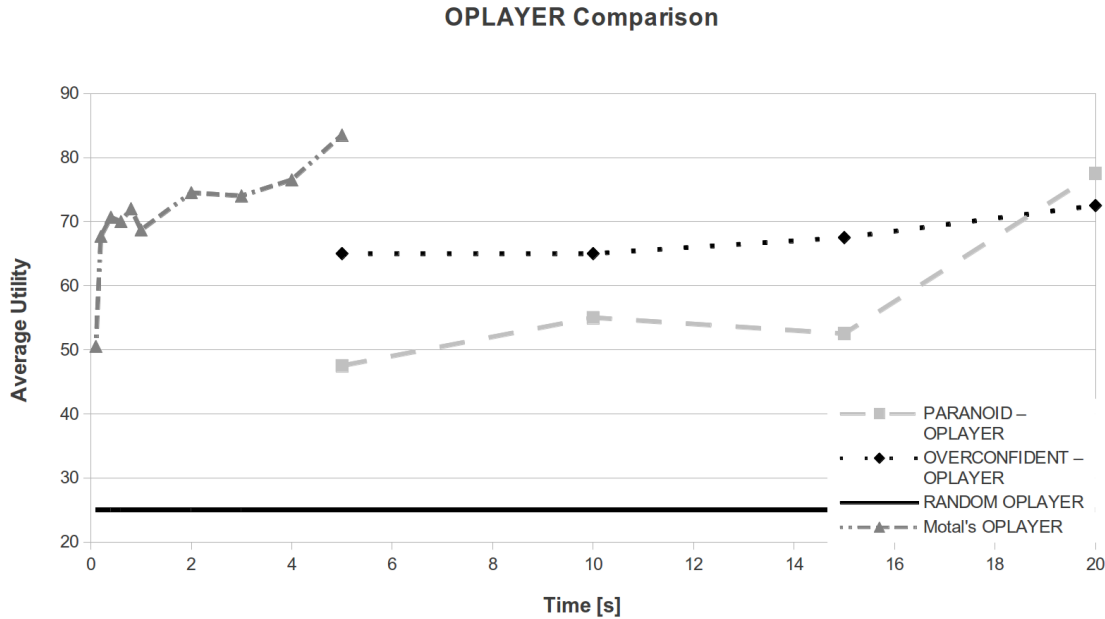


Figure 6.2: Comparison of Paranoid and Overconfident model with Motal's player at "O" position

Last comparison shows averaged results on both positions. As follows from results discussed above the Overconfident opponent model is better when playing against random player. Although the Paranoid also did well especially playing as XPLAYER it still wasn't enough to compensate much worse results when playing as the second. Unfortunately since we weren't able to run the fourth set of matches for Overconfident model at first position we present only three values however we are sure there will be similar result to the average utility of Paranoid model in 20 seconds per move.

The results of measuring are listed in tables 6.1 and 6.2 in the same pairs as they are in figures above. From this data one can observe the confidence interval is big which is caused mostly by small statistical sample.

<b>XPLAYER - Paranoid model</b>				
<b>time per move</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>
Average utility	90	95	92.5	95
Standard deviation	30	15	23.85	21.79
5-% Confidence interval	11.03	6.57	10.45	9.55
<b>XPLAYER - Overconfident model</b>				
<b>time per move</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>
Average utility	90	92.5	90	-
Standard deviation	25.5	23.85	25.5	-
5-% Confidence interval	11.17	10.45	11.17	-

Table 6.1: Numerical comparison of Paranoid and Overconfident models playing both as XPLAYER

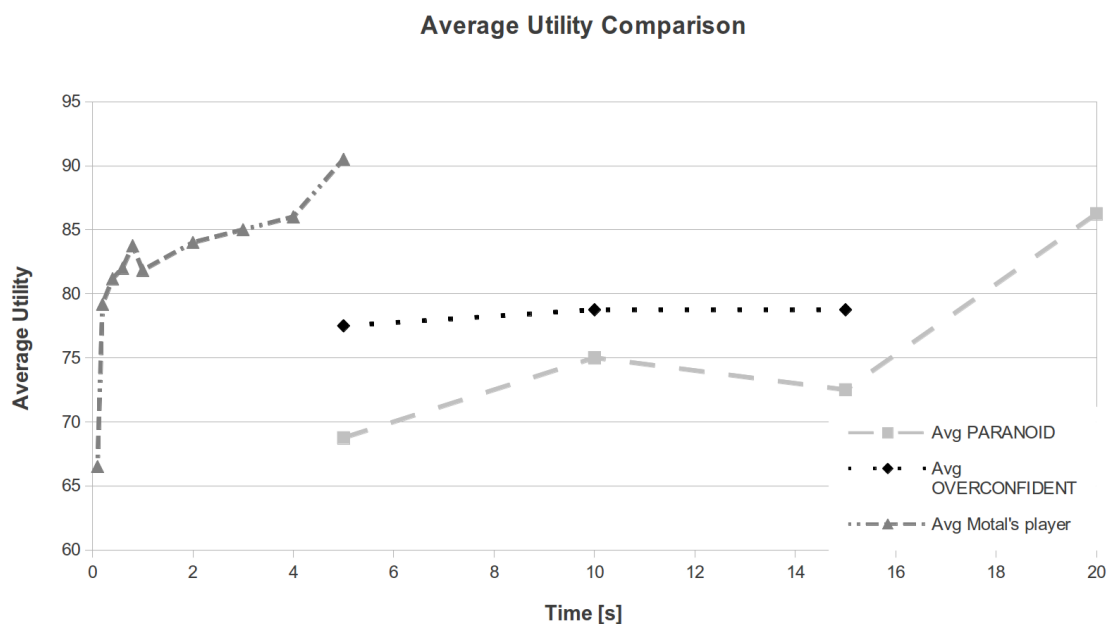


Figure 6.3: Comparison of Average Utility of Paranoid and Overconfident models on both positions with TIIGR player

<b>OPLAYER - Paranoid model</b>				
<b>time per move</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>
Average utility	90	95	92.5	95
Standard deviation	30	15	23.85	21.79
5-% Confidence interval	11.03	6.57	10.45	9.55
<b>OPLAYER - Overconfident model</b>				
<b>time per move</b>	<b>5</b>	<b>10</b>	<b>15</b>	<b>20</b>
Average utility	90	95	92.5	95
Standard deviation	30	15	23.85	21.79
5-% Confidence interval	11.03	6.57	10.45	9.55

Table 6.2: Numerical comparison of Paranoid and Overconfident models playing both as OPLAYER

In the end we would like to make small summary of results which we measured on Latent Tic-Tac-Toe.

First the TIIGR player did better than our player as it needed fewer time to make good moves. Second Overconfident played on average better than Paranoid which is caused by the precise opponent model of the former. Paranoid model performs better as XPLAYER since it plays better when Information Sets are smaller.

### 6.2.2 Simple card game

Second game we were testing ISS on is card game introduced together with GDL-II in [15]. Rules are following.

There are three players. First one is random player representing dealer the two others are common players. At the beginning of the game both players receive (from dealer) one of three cards - Jack, Queen or King whose values are in the increasing order and because it's IIG then both players don't see the opponent's card. Having their card both players have to actions they can make which are fold or bet. After both of them made their moves the following situations may occur.

- Both players bet. In such situation the one with the card of higher value wins. Winner gets utility 100 loser 0.
- One folds other bets. The one who bets receives utility 75 the one who folds obtains 25.
- Both folds. Both players gets utility 50.

ISS				TIIGR
time per move [s]	1	5	10	0.4
Average utility	60.25	62.25	64.75	57.68
Standard deviation	32.81	32.31	30.43	-
5-% Confidence interval	6.43	6.33	5.96	-

Table 6.3: Average utility with respect to the time per move in Simple card game and compared to TIIGR

Again we will make comparison with TIIGR player despite, as its author states, the time given for each turn was only 400ms. Because of the technical reasons we are unable to set time per move lower than 1 second.

Unlike the experiments with Latent Tic-Tac-Toe we ran 100 matches for each time setting. The data we measured are in table 6.3. However we can compare it only to one value which was measured in [8] which is 57.67. Since we don't have original data of each match we don't give deviation and confidence interval.

In contrary to the results in section 6.2.1 it's obvious our player performs much better compared to TIIGR than it does in mentioned section. Despite even Motal's player uses some kind of overconfident assumption as he states in his work as did ours in these matches the difference is smaller in contrast with matches in Latent Tic-Tac-Toe. There are at least two reasons why our player performs better. First the state space is very small so he is not handicapped by absence of caching and prevention for creating duplicate states. And second related to the previous we generate only very few observations which helps more than the size of state space as the generation is very expensive operation.

### 6.3 Tuning parameters

The two variables or parameters which we are going to focus on it this section are the number of simulations which one can run from expanded Information Set and the size of sample which we generate from IS which exceeds it.

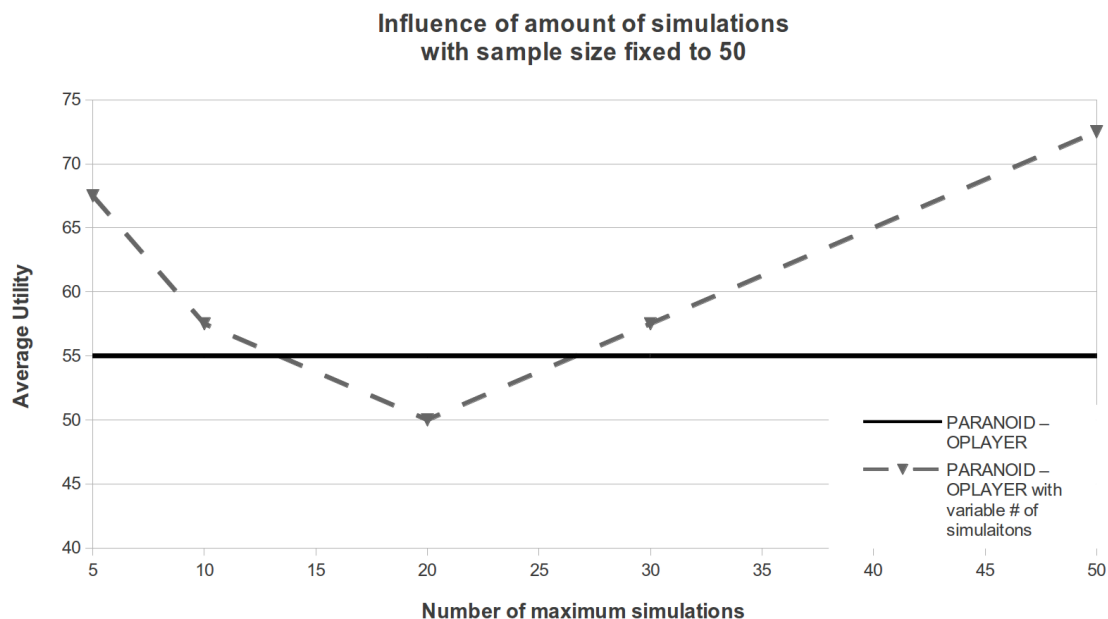


Figure 6.4: Influence of number of simulations on Average Utility

We'll describe how we arranged presented experiments. First the tests we made and which are described in this section uses Latent Tic-Tac-Toe. From section 6.2.1 we took ISS with Paranoid opponent model playing as OPLAYER and as a kind of referential point the average utility achieved with 10 seconds per move. All data was obtained by running 20 matches for each setting.

Let's remind that in Chapter 5 we mentioned these two variables aren't independent in our implementation since behaviour of Maximum of simulations depends on the Sample size. Before we proceed to the discussion about the results we need to explain this dependence first because the tests wouldn't make sense.

As was mentioned in the section 5.4 the variable representing the overall number of simulations the algorithm will make is taken as a minimum of the size of IS and Maximum of simulations variable. There the size of a sample needs to be taken into account.

Maximum of simulations	5	10	20	30	50
Average utility	67,5	57,5	50	57,5	72,5
Standard deviation	36,31	48,15	44,72	45,48	43,23
5-% Confidence interval	15,92	21,10	19,60	19,93	18,95

Table 6.4: Dependency of Average utility on Maximum of simulations with fixed sample size to 50.

Now we can finally start with discussion. The Figure 6.4 shows how the number of simulation which we ran influenced Average Utility. The overall trend of increasing Average Utility with the increasing amount of simulations is quite logical. As we already

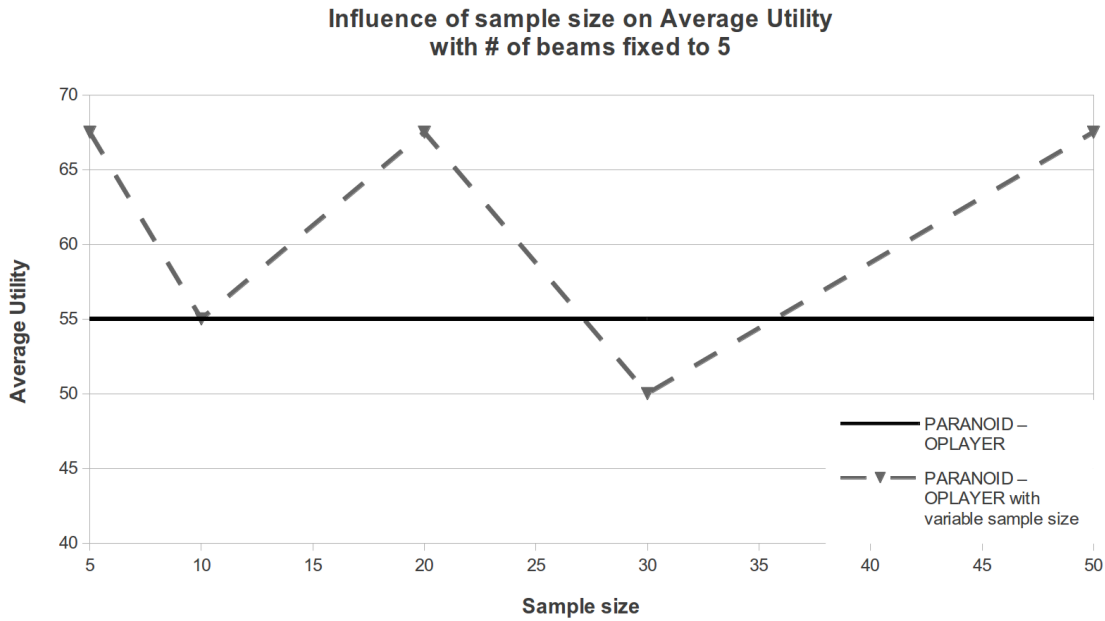


Figure 6.5: Influence of Sample size on Average Utility

wrote in Chapter 5 the generation of observations is expensive operation and it may occur the algorithm would spend the most of all time it was given just with the generating and this variable is good way on how to control the trade-off. As for the local minimum one need to realize the values aren't influenced only by the discussed variable but with the size of sample too as is stated in the previous paragraph.

Size of sample	5	10	20	30	50
Average utility	67.5	55	67.5	50	67.5
Standard deviation	42.65	41.53	39.61	47.43	36.31
5-% Confidence interval	18.69	18.20	17.36	20.79	15.92

Table 6.5: Dependency of Average utility on sample size with fixed number of simulations to 5.

In the second set of values we set the number of simulation to 5. If we wouldn't do this there would occur such situation which follows from dependence mentioned at the beginning of this section i.w. situation in which the matches which have Sample size set higher or equal to selected amount of simulations.

Unlike the previous one the Figure 6.5 isn't much clear. It appears the only reasonable explanation of what happens is that due to the smaller number of matches which we run some of averaged results may not represent real value which it should converge on. The results of both experiments are in tables 6.4 and 6.5.

# Chapter 7

## Conclusion

### 7.1 Summary

Literary the foundations of our work consists of two algorithms. The first of them is MCTS. Simply said it's algorithm designed for state space search based on random passes through the game tree from the root to the terminal states. The second - Minimax based ISS which is unlike MCTS intended for IIG playing. Insead of paranoid assumption which is present in Minimax it uses so called opponent models by which it tries to represents opponent's play. By combination of these two algorithms we get ISS implemented as MCTS and that is the crucial point of our work.

For ISS implementation we have chosen Palamedes framework which is designed specially for GGP. Together with framework itself there are also bundled so called provers whose task is to transform the games from the textual representation to the form which can be easily used by any state space search algorithm and enables him to perform all operations necessary for sucessful implementation. Unfortunately the Palamedes exists only in version for FIG. With the usage of already implemented functions and classes for Javaprover and several existing functions for more high-level manipulation with prover we extended Palamedes to support IIG. The extension was written general enough to enable implementation of other algorithms than ISS.

Important aspect of IIG playing is good representation of IS. We discuss three different approaches - Game decomposition, State agregation and Statistical sampling. From them we selected Statistical sampling and used it in ISS because it is easy to implement and doesn't require any additional changes in Javaprover.

We compared indirectly our ISS implementation with TIIGR player in games Latent Tic-Tac-Toe and Simple card game. Here indirectly means we let our player run against random player and then we compared our results with the results of TIIGR player. In the first mentioned game the results definitely showed the TIIGR performed better. On the other hand in the card game the results were approximately equal. Based on the comparison of the state space size of these two game and based on the results we can say our player performed worse most likely due to duplicit state generation which occurs in Palamedeses extension.



## 7.2 Future work

Our future work may concern especially the following items which are ordered according to the importance from our point of view.

1. Enhancement of our extension. As we already mentioned the player we implemented suffers from the absence of caching which may result in significant decrease of performance. In future versions of the extension we would like to add the caching.
2. There exists several variables which can enhance the performance of ISS. We would like to test the performance of player using these variables.
3. Since the speed of Javaprover isn't the best we could implement the support of GDL-II into Prologprover.

## Chapter 8

# Bibliography

- [1] G. Chaslot, M. Winands, J. van den Herik, J. Uiterwijk, and B. Bouzy. Progressive strategies for monte-carlo tree search. In *Joint Conference on Information Sciences*, 2007.
- [2] Paolo Ciancarini and Gian Piero Favini. Monte carlo tree search in kriegspiel. *Artificial Intelligence*, 174(11):670 – 684, 2010.
- [3] Hilmar Finnsson. Cadia-player: A general game playing agent. Master’s thesis, Reykjavík University - School of Computer Science, 2007.
- [4] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aai competition. *AI Magazine*, 26(2):62–72, 2005.
- [5] Ingo Keller. Palamedes a general game playing ide. Master’s thesis, TU-Dresden, 2009.
- [6] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML-06*, 2006.
- [7] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General game playing: Game description language specification. Technical report, 2008. most recent version should be available at <http://games.stanford.edu/>.
- [8] Tomáš Motal. General game playing in imperfect information games. Master’s thesis, FEE CTU in Prague, 2011.
- [9] Jean Méhat and Tristan Cazenave. A parallel general game player. *KI*, 25(1):43–47, 2011.
- [10] Austin Parker, Dana Nau, and VS Subrahmanian. Overconfidence or paranoia? search in imperfect-information games. *AAAI*, 2010.
- [11] Jacques Pitrat. Realization of a general game-playing program. In *IFIP Congress*, pages 1570–1574, 1968.
- [12] Eric Rasmusen. *Games and Information: An Introduction to Game Theory*. Blackwell Publishers, 2006.

- [13] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1191–1196. AAAI Press, 2007.
- [14] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems*. Cambridge University Press, 2009.
- [15] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 994–999. AAAI Press, 2010.
- [16] Dengji Zhao. Decomposition of multi-player games. Master’s thesis, TU-Dresden, 2009.

# Appendix A

## List of Abbreviations and Acronyms

**EU** - Expected utility

**EFG** - Extensive-form game

**GDL** - Game Description Language

**GGP** - General Game Playing

**IIG** - Imperfect-information game

**IS** - Information set

**ISS** - Information set search

**PIG** - Perfect-information game

**MCTS** - Monte Carlo tree search

**UCT** - upper Confidence bound applied to trees