University of Alberta

SIMULTANEOUS MOVE GAMES IN GENERAL GAME PLAYING

by

Mohammad Shafiei Khadem

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2010
Edmonton, Alberta

Examining Committee

Jonathan Schaeffer, Computing Science

Nathan Sturtevant, Computing Science

Bora Kolfal, Accounting and Management Information Systems, School of Business

Duane Szafron, Computing Science

Abstract

General Game Playing (GGP) deals with the design of players that are able to play any discrete, deterministic, complete information games. For many games like chess, designers develop a player using a specially designed algorithm and tune all the features of the algorithm to play the game as good as possible. However, a general game player knows nothing about the game that is about to be played. When the game begins, game description is given to the players and they should analyze it and decide on the best way to play the game.

In this thesis, we focus on two-player constant-sum simultaneous move games in GGP and how this class of games can be handled. Rock-paper-scissors can be considered as a typical example of a simultaneous move game. We introduce the CFR algorithm to the GGP community for the first time and show its effectiveness in playing simultaneous move games. This is the first implementation of CFR outside the poker world. We also improve the UCT algorithm, which is the state of the art in GGP, to be more robust in simultaneous move games.

In addition, we analyze how UCT performs in simultaneous move games and argue that it does not converge to a Nash equilibrium. We also compare the usage of UCT and CFR in this class of games. Finally, we discuss about the importance of opponent modeling and how a model of the opponent can be exploited by using CFR.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

For many years artificial intelligence researchers built specialized game players that focused on the complexities of a particular game. For example Deep Blue and Chinook play the game they are built for, *viz.* chess and checkers respectively, at an expert level [6, 30].[1] However, part of their success is due to the knowledge engineering that has been done by their developers. Therefore most of the interesting game analysis task is done by humans instead of the programs themselves. The idea of using game playing to evaluate progress in AI and pushing game analysis toward the program instead of the human resulted in the concept of *General Game Playing* (GGP). GGP is the idea of having a player capable of playing any game that is described using a predefined *Game Description Language* (GDL). Since general game players do not know the game they are going to play in advance, there cannot be specialized algorithms for solving a particular game hardcoded in their nature. However, to perform well they should incorporate various AI technologies such as knowledge representation, reasoning, learning, and rational decision making. Barney Pell first suggested in 1993 the differentiation between whether the performance of an agent in some game is due to the general success of the AI theories it embodies, or merely to the cleverness of the researcher in analyzing a specific problem in [27]. He introduced a variant of GGP under the name of "Metagame" which addressed the domain of symmetric chess-like games [27]. He also proposed a Prolog-like game description language for defining the game logic.

In this chapter we will first consider the GGP competition and the characteristics of the games used in the competition in Section 1.2. We will then review the Game Description Language used for describing the games in the GGP competition in Section 1.3 and the gaming infrastructure in Section 1.4. In Section 1.5 we define what we mean by *simultaneous move games*, the class of games that we will focus on in this thesis. We detail the contributions of this thesis in Section 1.6.

[1] Chinook is a perfect checkers player that plays at superhuman level.

1.2 GGP Competition and Games' Characteristics

To encourage research in GGP, there has been an annual competition at the AAAI (Association for the Advancement of Artificial Intelligence) conference since 2005 with a $\$10,000$ grand prize. All the games in the competition are guaranteed to be *finite*. They all have a finite number of states, one distinguished start state, and a finite set of terminal states. All the games are also *synchronous*, meaning that all the players take an action at every step simultaneously (often allowing "no-operation" as an action choice). Every game must also be discrete, deterministic, complete information, playable, and weakly winnable (or strongly winnable in single player games). Playability and winnability are defined as follows in the game description language specification [26].

Definition 1 *A game is **playable** if and only if every player has at least one legal move in every non-terminal state.*

Definition 2 *A game is **weakly winnable** if and only if, for every player, there is a sequence of joint moves of all players that leads to a terminal state of the game where that player's goal value is maximal.*

Definition 3 *A game is **strongly winnable** if and only if, for some player, there is a sequence of individual moves of that player that leads to a terminal state of the game where the player's goal value is maximal.*

An abstract definition of the games used in the GGP competition is given in [16] using *finite state machines* (FSM).[2] The components of the FSM describing a game are the following as described in [16]:

- S*, a set of game states.* This is the set of states of the FSM (Q).
- $r_1, \ldots, r_n$*, the* n *roles in an n-player game.*
- $A_1, \ldots, A_n$*,* n *sets of actions, one set for each role.*
- $l_1, \ldots, l_n$*, where each* $l_i \subseteq A_i \times S$. These are the *legal* actions in a state where each l_i defines the legal action of the i^{th} player in all the states of the game. Every $(a_1, \ldots, a_n) \in l_1 \times \ldots \times l_n$ is a member of the FSM alphabet (Σ).

[2] Due to Sipser [35], a **finite state machine** or **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.

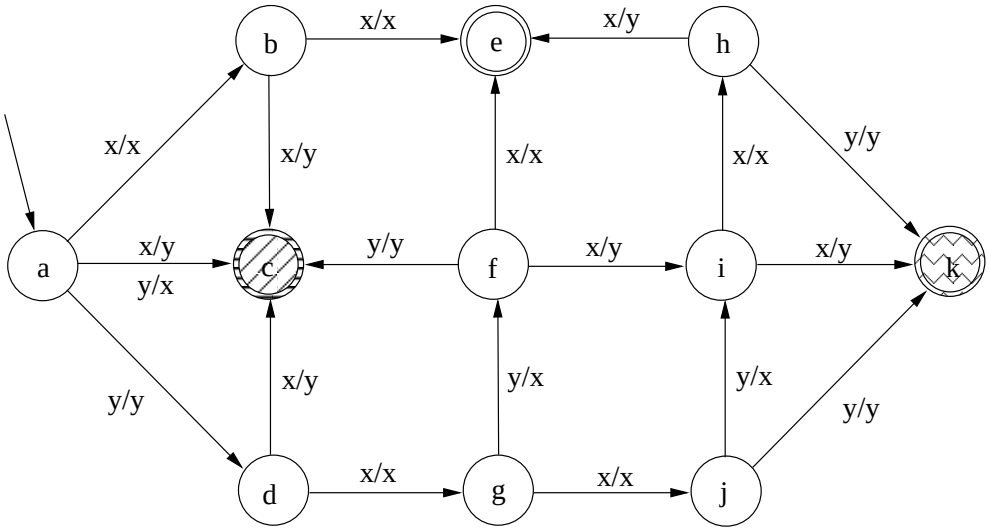


Figure 1.1: The FSM representation of a game adapted from [16].

- $\eta : A_1 \times \ldots \times A_n \times S \rightarrow S$*, an update function mapping.* This is the FSM transition function defining how the transitions between different states are made according to the actions being selected by different players (δ).
- $s_0 \in S$*, the initial game state.* This is the start state of the FSM (q_0).
- $g_1, \ldots, g_n$*, where each* $g_i \subseteq S \times [0 \ldots 100]$. Every g_i defines the outcome for the i^{th} player.
- $T \subseteq S$*, a set corresponding to the terminal states of the game.* This is the set of accept states of the FSM (F).

Figure 1.1 shows the FSM representation of a two player game. In this game, we have $S = \{a, b, c, d, e, f, g, h, i, j, k\}$, $s_0 = a$, and $T = \{c, e, k\}$. Different shadings of c and k indicate that these states are the significant states, *i.e.* winning states, for different players. Transitions between states are shown with arrows that are marked with the respective action chosen by different players (first-player/second-player). Each player has two different actions, *viz.* $\{x, y\}$, which are not legal in all the states. For example the first player can only take action x in state d, while both of his actions are legal in state f. Using the FSM representation, required properties of the games can also be checked.

- For a game to be finite, no cycles can exist in the FSM graph.
- Game playability requires that there is at least one outgoing edge from every non-terminal state.
- Game winnability requires that for every player, there is a path in the FSM graph from the starting state to the significant state of that player where the player's goal value is maximal. If

this path only depends on the player's own actions then the game is strongly winnable for that player. Otherwise the game is a weakly winnable game for that player.

It should be noted that the FSM description is very similar to the traditional extended normal form description in game theory. However, the FSM representation uses a graph representation while the extended normal form uses a tree representation and the outgoing edges at each node are the actions of the player who has to move at that node. The FSM representation fits the synchronous nature of the games in GGP better than the sequential tree representation of the extended normal form. However, the extended normal form can also be used for describing a game. We will use the tree representation in the following chapters.

1.3 Game Description Language

The *Game Description Language* (GDL) in the GGP competition is a variant of *Datalog* that allows function constants, negation, and recursion [26]. The *Knowledge Interchange Format* (KIF) is used for GDL representation. GDL describes the game state in terms of a set of true facts. The transition function between states is described using logical rules in GDL that define the set of true facts in the next state in terms of the current state and the actions performed by all the players (the *next*[3] relation is used for this purpose). GDL also contains constructs for distinguishing between initial states as well as goal and terminal states. Items in the GDL are either facts or implications. Facts include role definition, initial state description, and game specific facts. Implications can be action definitions, action effects, axioms, goal or terminal state description, and game specific relations.

Each game description is a set of terms, relational sentences, and logical sentences which contain a set of variables, constants, and logical operators. Language keywords are as follows:

- **role(**$< a >$**)** means that $< a >$ is a role (player) in the game.
- **init(**$< p >$**)** means that the datum $< p >$ is true in the initial state.
- **true(**$< p >$**)** means that the datum $< p >$ is true in the current state.
- **does(**$< r >, < a >$**)** means that player $< r >$ performs action $< a >$ in the current state.
- **next(**$< p >$**)** means that the datum $< p >$ is true in the next state.
- **legal(**$< r >, < a >$**)** means it is legal for $< r >$ to play $< a >$ in the current state.
- **goal(**$< r >, < v >$**)** means that player $< r >$ would receive the goal value $< v >$ in the current state.
- **terminal** means that the current state is a terminal state.
- **distinct(**$< p >, < q >$**)** means that the datums $< p >$ and $< q >$ are syntactically unequal.

```
1.  (role xplayer)
    (role oplayer)

2.  (init (cell 1 1 blank))
    (init (cell 1 2 blank))
    (init (cell 1 3 blank))
    (init (control xplayer))

3.  (successor 1 2)

4.  (<= (legal ?p (mark ?x ?y))
        (true (cell ?x ?y blank))
        (true (control ?p)))

5.  (<= (next (cell ?m ?n x))
        (does xplayer (mark ?m ?n))
        (true (cell ?m ?n blank)))

6.  (<= (next (cell ?m ?n blank))
        (does ?p (mark ?j ?k))
        (true (cell ?m ?n blank))
        (or (distinct ?m ?j) (distinct ?n ?k)))

7.  (<= (next (control oplayer))
        (true (control xplayer)))

8.  (<= open
        (true (cell ?m ?n blank)))

9.  (<= (goal xplayer 100)
        (line x))
    (<= (goal xplayer 50)
        (not (line x))
        (not (line o))
        (not open))

10. (<= terminal
        (line x))
```

Figure 1.2: Sample GDL code from tic-tac-toe game description. Words with Boldface font are keywords.

Figure 1.2 shows a sample of GDL code from the *tic-tac-toe* game description. The relations in part 1 define the roles of the games, which are *xplayer* and *oplayer* in this game. The initial state is defined using the *init* keyword. For example in the initial state in Figure 1.2, cells are blank at first and *xplayer* is in control (relations in part 2 of Figure 1.2). There can also be game related facts in the game description similar to part 3 in Figure 1.2.[4] Actions are defined using the *legal* keyword. For each action the preconditions for taking that action are also defined. It is defined in Figure 1.2 part 4 that a player can *mark* a cell if it happens to be blank and the player is in control of the game at that step. The names which start with a question mark in the GDL are variables. The effects of taking an action are defined using the *next* relation. In part 5 of Figure 1.2, it is stated that a cell will be marked with x if it happens to be blank and then marked by *xplayer*. The axioms of the world are also defined using the next relation (*e.g.* part 6 of Figure 1.2). For example part 6 of Figure 1.2 indicates that a cell remains blank if it is blank and no player marked that cell. Game related relations can also be defined in a game similar to part 8 of Figure 1.2 that says the board is still open as far as there are blank cells. Goal and terminal states are defined using *goal* and *terminal* keywords respectively (*e.g.* parts 9 and 10 in Figure 1.2). Interested readers are referred to [26] for a comprehensive discussion about Datalog as well as the GDL used in the GGP competition. A complete description of tic-tac-toe in GDL is given in Appendix A.

1.4 Game Management Infrastructure

In the GGP competition at the AAAI conference there is a *Game Master* (GM) which is responsible for providing players with the game description and ensuring that each player takes a legal action at each step. Each game consists of two phases, setup and play. At the beginning of each game, the description of the game is given to players using GDL, and players have a specific amount of time, called the *start-clock*, to analyze the game structure and set up any infrastructure they will need during the game. After the set up phase and during the playing phase, each player has limited amount of time, called the *play-clock*, to submit its action at each step. If a player fails to submit a legal action by the end of play-clock, the GM considers a random legal action as the action chosen by the player. After each step the GM informs all players about all actions that were performed by all players. All the communications between the GM and players are over TCP/IP connections. The GM is responsible for checking that the state of the game and moves submitted by all the players are legal at all steps of the game. A schematic view of the competition setup is given in Figure 1.3.

[3] *next* is a keyword in GDL.

[4] These three groups were the facts in code segment given in Figure 1.2. The rest of the code segment consists of implications.

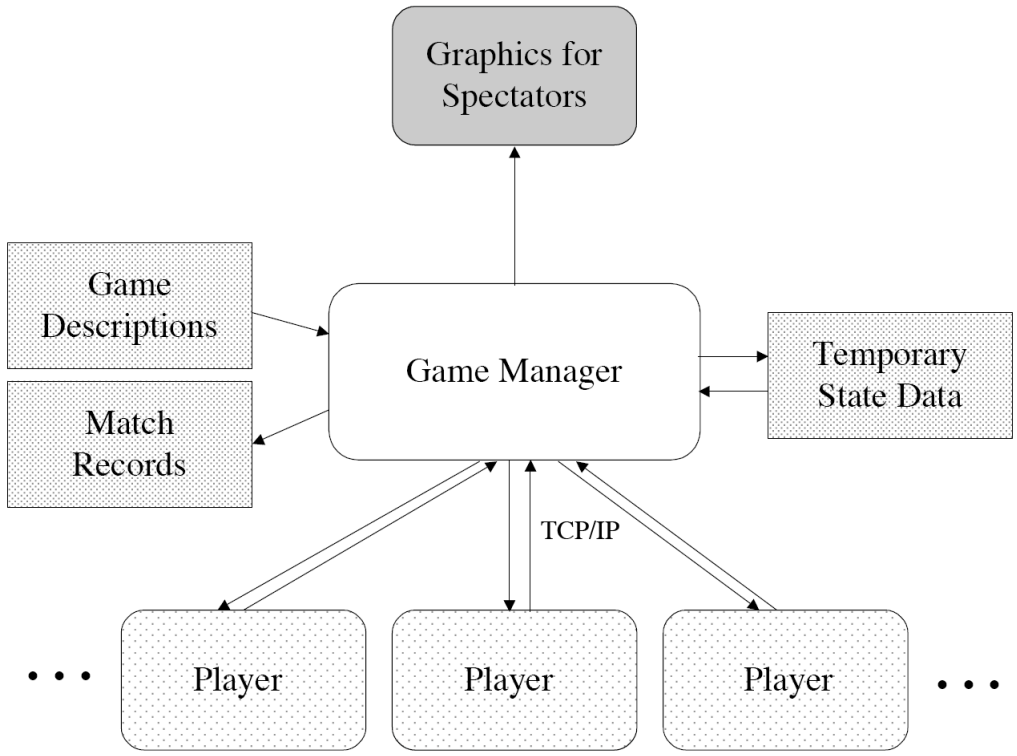


Figure 1.3: Schematic view of GGP competition from [16].

1.5 Simultaneous Move Games

As stated in Section 1.2, all the games in the GGP are *synchronous* and require all the players to take an action at every step simultaneously. Turn taking games are described in GGP by defining a *no-operation* for the player who does not have control. Therefore, although every game is described as a simultaneous game between all the players, not all of them are necessarily simultaneous move games. We consider the following definition for *simultaneous move games*.

Definition 4 *A multiplayer game in the GGP is a* ***simultaneous move game*** *if and only if, there is at least one time in the game when two or more players can alter the state of the game on the same move.*

The class of games that we deal with in this thesis are two-player constant-sum simultaneous move games. We will consider the computation of Nash equilibrium in this class of games, which we will discuss why it is reasonable in Chapter 3 Section 3.6.1. Our focus will be on two-player constant-sum simultaneous move games because of the fact that in multi-player general-sum games, the final Nash equilibrium is not well-defined and can be different based on the assumptions that one player can make about other players, *e.g.* whether other players make a coalition against us or not. From now on, whenever we mention simultaneous move games we mean two-player constant-sum simultaneous move games unless clearly stated.

1.6 Thesis Overview and Contributions

In Chapter 2 we present previous work in the GGP domain. In Chapter 3 we analyze the UCT algorithm, which is the state of the art algorithm used in GGP. We explain how it fails in simultaneous move games. We consider enhancements that improve the quality of UCT in simultaneous move games and make it more robust. However, we show that even the enhanced version of UCT will not converge to the Nash equilibrium in general and the solution that it converges to is brittle. We describe the CFR algorithm in Chapter 4 and consider how it can be used in GGP. We implemented CFR to be used in GGP. The drawbacks of using plain CFR and the need for opponent modeling is discussed in Chapter 5. Finally, we argue how the CFR algorithm can be used to exploit the UCT algorithm. Experimental results for each of the aforementioned sections are also provided. We will wrap the thesis with a discussion of the challenges for playing simultaneous move games and definition of trends for future work.

The major contributions of this thesis are as follows.

1. We provide an implementation of a general game player that uses CFR to play the games. This is the first time that CFR has been used in GGP and it demonstrates that CFR is a general algorithm and can be widely used in domains other than poker, which it was developed for at first. In addition, we will discuss in Chapter 4 that using CFR in GGP is more convenient than its usage in poker because all the games in GGP are deterministic.

2. We show that the performance of the CFR player is robust. We also show that there are situations where the performance of the CFR player is better than an enhanced UCT player.

3. We discuss the importance of opponent modeling and how CFR can be used to exploit a known model of an opponent. We show that the solution that UCT converges to in simultaneous move games can be exploitive and this situation can be easily exploited by CFR.

1.7 Summary

In this chapter we introduced General Game Playing (GGP) as well as game characteristics and features. We also briefly reviewed the Game Description Language as well as the infrastructure used in the AAAI competition. We also defined the class of games that will be our main focus and briefly reviewed our contributions.

Chapter 2

General Game Players

2.1 Introduction

A general game player is an agent that interacts with its environment without having prior knowledge about it. At the beginning of each match, each player receives the game description and must play as well as possible while adhering to the rules of the game. It is challenging to develop such an agent because the specification of games that an agent is going to face is not known beforehand. Therefore designers cannot simply take advantage of unique features of a particular game by hardcoding them into their players. This means if you are a chess master, your player will not necessarily play chess well unless you can detect that a game is actually chess. A general game player must read a game description in GDL and extract game features from it and try to play well either alone or in collaboration with teammates and possibly against opponents according to the game rules.

Different approaches have been used to create general game playing programs. In this chapter we review different approaches that have been tried in development of the general game players as well as how different researchers have tried to improve the quality of GGP programs.

2.2 Early Work on General Game Players

The first general game player was the *Metagamer* that Pell developed to handle *Metagame* in Symmetric Chess-Like games (SCL-Metagame) [27]. His player is more important from the historical point of view than the technical perspective. The player had a search engine based on the *minimax* algorithm with alpha-beta pruning. The search engine was guided using a heuristic generator. The heuristic generator used a combination of general features defined by Pell (the human designer) that he considered to be useful to look at in the class of SCL-Metagames. These features included *mobility*, *centrality*, and *promotion* that were trying to capture the following information about the game.

- Mobility was a notion of the number of moves available to the player. It was assumed that having more moves is generally better because it gives the player more options to choose

from.

- Centrality gave bonus to positions with more pieces at the center of the board or closer to the center. The idea was that more centrality will increase mobility as central cells on a chess board have access to all the directions if they are not blocked. In addition, the longest shortest path to any other cell is shorter from a more centralized cell. Therefore a player can get to different parts of the board in fewer number of steps.

- Promotion accounted for how different pieces on the board can be promoted to new pieces.

A weighted sum of the combination of the features was then used by the heuristic generator as the final heuristic. Pell defined the weights for each feature manually and set up a tournament between players with different weight settings to show how they would perform.

2.3 The First Generation of General Game Players

The announcement of a new GGP competition in 2005 drew attention of many researchers toward the development of general game players. Initially, most program developers tried to develop heuristics for a game by extracting features from the game description [8, 32, 23, 19]. However, in contrast to Pell's approach which had SCL-Metagame features encoded in the program, they were trying to extract useful features of the game from the game description. The heuristics were then used in conjunction with a classic search algorithm (*e.g.* alpha-beta) to play the game. Therefore, devising a good heuristic was a key factor in the success of their approaches. However, inventing a good heuristic is a challenging problem since the game that is going to be played by the player is unknown beforehand.

Clune tried to extract features by analyzing the game description, considering the stability of various components [8]. The idea was that the components that do not change drastically from a state to the next state are the important features of the game. He then used those features to build an abstract model. The final heuristic was then built using the cost of solving the problem in the abstract model. His approach was heavily dependent on his feature extraction and the abstract model being built. Clune implemented his ideas in a player called *Cluneplayer* and participated in several GGP competition, winning the 2005 competition and placing second in the 2006 and 2008 competitions.

Schiffel *et al.* also used structure detection to extract game features, *e.g.* a board, from the game description [32]. In addition they utilized the percentage of goal satisfaction as a metric to derive heuristic to search the state space and bias their search toward states that most likely would satisfy the logical description of the goal of the game. They used fuzzy logic in their computation. They computed a ratio of truth for the atoms of the goal and assigned a truth value to the goal based on a set of predefined rules. However, the problem of implicit goal description could not be addressed in their approach. Schiffel *et al.* implemented their ideas in a player called *Fluxplayer* and participated in several GGP competition, winning the 2006 competition.

Kuhlmann *et al.* tried to extract features of a game by assuming some syntactic structure in the GDL [23]. They verified the resulting features by running a number of internal simulations. If the features held throughout the simulations, they assumed that the extracted features were correct. However, their approach was brittle facing an adversarial game description, since it solely relied on the syntactic structure of the GDL. For example, they assumed that a successor relation is a relation with a name and arity of two.[1] With this assumption if just a dummy variable is added to the successor relation, then it cannot be defined as a successor relation anymore using their approach. In addition, they statically chose how different features would be weighted in a game. Therefore, based on whether a feature must be weighted positively or negatively in a game, their weighting could be wrong. Kuhlmann *et al.* implemented their ideas in a player and participated in several GGP competitions, but never won or played in the finals.

Kaiser tried to extract features, *e.g.* pieces and motions[2] in games, by computing the variance of the parameters of the relations and comparing states generated from random plays of the game [19]. He built some heuristics using extracted knowledge, *e.g.* distance between states and to the target, piece counts, etc. He searched the state space using the derived heuristics to reach the goal. However, his approach was limited by the number of plays that he could run and how well he could explore the state space. Games whose states were not fully described at each step were difficult to handle using his approach as well.

All the players that are based on classical search methods use a heuristic function derived from the game description. The heuristic is a critical part of these players. However, deriving a good heuristic by analyzing the game description is a complex task. In addition, players are given a scrambled version of the game description during the competition, to prevent any human intervention in interpreting the game description. Obtaining a good heuristic only by relying on the game structure is a hard task to accomplish as well because the game description can also be adversarial by containing useless rules and variables included in the game description to fool the player. Although extracting features and detecting objects in the game description is a complicated task to be achieved, but players based on classical search methods that use these techniques are still successful and among the top players (*e.g.* Fluxplayer and Cluneplayer).

2.4 Monte-Carlo Based Players

After the success of Monte-Carlo search methods in Go [14], researchers started investigating its usage in GGP. Go is a strategic two-player turn-taking board game. It has simple rules, but has a large branching factor. In addition, no efficient evaluation function approximating the minimax value of a position is available [14]. Therefore, using Monte-Carlo simulation to evaluate the value of a position in a match paid off and *MoGo*, which is a Monte-Carlo based program, is currently one

[1] (successor first second)

[2] Motion detection is concerned with detecting the pieces that move around and how they move.

of the strongest Go players [14].

Monte-Carlo based methods need no specially designed evaluation function at leaf nodes of the game tree, because random simulations are used to acquire a value. This feature greatly simplifies the design of general game players, since game description analysis is no longer mandatory. However, a player with a better understanding of the game description can exploit that knowledge and be a better player by using it during simulation and doing a better trade off between exploration and exploitation. UCT is a fairly simple and effective Monte-Carlo tree search algorithm [22]. It is a kind of best-first search which tries to balance deep searches of high-winrate moves with exploration of untried moves. Since there is no need for an evaluation function in Monte-Carlo based methods, UCT was a natural choice for use in GGP. A thorough discussion of UCT will be given in Chapter 3.

Hilmar and Björnsson developed the first Monte-Carlo player, called *CadiaPlayer* [12]. Starting in 2007, they used the UCT algorithm to play every game regardless of whether it is a single-agent, two-player constant-sum, two-player general-sum, or multi-player game [12]. They became the champion in the first year (2007) that they entered the competition, and with some minor changes to their program, they also won in 2008 [5]. That they were able to won with a "knowledge-free" solution demonstrates how hard it is to extract knowledge from the game description and build a useful heuristic to be used in conjunction with classical search methods. After the successful appearance of CadiaPlayer, many more contestants were drawn toward using UCT in their players.

2.5 Using Reinforcement Learning for General Game Playing

Levinson performed an early investigation of Reinforcement Learning (RL) [37] for GGP[3] in 1995 and also reviewed different approaches for the development of a general game player [25]. Levinson used conceptual graphs in his work. He discussed a model called *Morph*, which was an application of adaptive-predictive search methods for improving search with experience. He also proposed an improvement over the stated original Morph model and demonstrated usage of its improved version in the development of an RL general game player.

Asgharbeygi *et al.* developed a relational temporal difference learning agent for GGP [1]. Their agent was composed of an inference engine, a performance module, and a learning element. It also used an external state transition simulator. Their inference engine generated all next states and their performance module decided which action to take. The learning element interacted with the simulator and performed relational temporal difference updates on utilities, which in turn controlled the state-value function. They also exploited the relational structure of game descriptions as well as attempted to generalize across different ground literals for each predicate.

Defining initial values on which an RL general game player should base its action selection policy is an issue that Banerjee *et al.* addressed in their work in 2007 [4]. They proposed a way

[3]The GGP that Levinson considered was different from the AAAI GGP competition.

of transferring value-functions from the games previously played by the agent to a new game, by matching states based on a similarity measure in an abstract representation, called *game independent features*, using game-tree lookahead structure. They claimed that the agent learns faster when it uses knowledge transfer and also can exploit opponent weaknesses if it faces the same opponent in future. The main advantage of the Banerjee *et al.* work over the Asgharbeygi *et al.* work was that it did not need to define game-specific or opponent-specific features for the transfer to be successful.

Banerjee *et al.* did similar work in value function transfer for GGP in earlier work [3]. In their former work they used a Q-learning RL agent. Most of ideas were similar in both works, but they used a handcrafted game-tree structure in the former and also proposed a method for defining board games and taking advantage of symmetry in those games. They also compared their agent benefiting from feature transfer against an agent using symmetry transfer, and showed that feature transfer outperforms symmetry transfer agent. In the latter work they did a better abstraction of game specific features and transferred game-independent features to a wider range of new games.

Kuhlmann *et al.* performed transfer learning using a graph-based domain mapping that constructed a mapping between source and target state spaces in GGP [24]. They first represented game description in GDL using a directed graph, called *rule graph*, and then used isomorphism over rule graphs to define identical games. They also generated variants of games played before and checked them against the new game through rule graph isomorphism to find any similarities. In their approach, if the new game was identical to the previously played game, then they would take full advantage of it, but there was no difference between similar and identical games in the Banerjee *et al.* work. Kuhlmann *et al.* transfered the state-value function versus the state-action value function used by Banerjee *et al.*.

Although the RL approach may seem promising, no general game player using RL has entered the competition and shown reasonable performance yet.

2.6 Other Approaches to GGP

In addition to the approaches for developing general game players that have been reviewed, there have been other approaches considered.

Reisinger *et al.* suggested the use of a coevolutionary approach to evolve populations of game state evaluators that can be used in conjunction with game tree search [28]. Their approach was very simple. They started with a population of neural networks which had state variables as the inputs and evaluated value of the state as the output. They tried to update the neural networks by adding/removing connections and changing weights to come up with a good game state evaluators. They did the mapping for the inputs randomly and used random players evolving simultaneously through coevolution to evaluate their game state evaluator population during coevolution.

Sharma *et al.* considered the use of an *Ant Colony System* (ACS) to explore the game space and evolve strategies for game playing [34]. In their approach they assigned different roles to each ant

and tried to update the level of play by updating the knowledge of ants through number of iterations. However, they did not considered the amount of time needed to train the ants, therefore it is hard to tell how their approach was.

Kissmann *et al.* tried to fill the gap between the planning and GGP community by trying to generate an instantiated game description in a format close to a planner input [21]. They generated an output similar to PDDL that could be used in any player or solver that needed the game description in instantiated form. However, their instantiating process seems time consuming and the resulted instantiated game description can be huge based on the number of objects in the game and arity of the relations.

2.7 Improving the Level of GGP

In addition to the effort that has been put in developing general game players, there is work that has investigated how to improve the performance of general game players. Work has been done in the context of knowledge representation, reasoning, knowledge extraction, and so on. Kirci *et al.* successfully applied learning based on state differences [20]. They learnt both defensive and offensive features. Finnsson *et al.* compared four different approaches for controlling Monte-Carlo tree search [13]. These approaches differed in how the back-propagation and updates are done. Schiffel *et al.* tried proving features in GGP using Answer Set Programming [33]. Schiffel also considered detecting symmetries in a game using GDL [31]. In addition, work has been done to simplify game descriptions by breaking it into smaller games if it is possible. Cox *et al.* and Gunther *et al.* considered game factoring and game decomposition [10, 17].

2.8 Conclusion

In this chapter we reviewed different approaches to the development of general game players and considered their advantage and disadvantage. In addition we considered how different researchers have tried to improve the quality of GGP by trying to focus on a specific aspect of the GGP.

Chapter 3

Universal Solution: the UCT Algorithm

3.1 Introduction

In this chapter we will consider and analyze the UCT algorithm in simultaneous move games. We will briefly review the background literature on the UCB algorithm that UCT is based on (section 3.2). We will describe the UCT algorithm (Section 3.3) and discuss its different properties (Sections 3.4.1 through 3.4.4). Finally we will consider the usage of UCT in simultaneous move games and its weakness and strengths (Section 3.6).

3.2 Background

The multi-armed bandit problem is an example of a problem where an agent tries to optimize his decisions while improving his information about the environment at the same time. If we consider a K-armed bandit, we are dealing with K different slot machines whose outcomes follow different unknown distributions with different expected values. Optimal play for a K-armed bandit is to select the arm with the highest payoff at each step of play. However, since we do not know the distribution of outcomes for different arms, the goal is to be as close as possible to the optimal play based on our past experiences. By careful tuning how much we exploit the best known arm versus exploring the other arms, we can bound the regret that results from selecting a suboptimal action (pulling the suboptimal arm). UCB1 (Upper Confidence Bound) is an algorithm that balances exploration and exploitation. It achieves a logarithmic bound in the number of plays on the expected regret after that number of plays [2]. It considers a bonus for selecting each action which is directly proportional to the number of plays and inversely proportional to the number of times that a specific action has been selected previously. Therefore, actions that have been rarely selected will get a higher bonus to be selected and explored.

3.3 UCT Overview

UCT (UCB applied to Trees) is an extension of the UCB1 algorithm to trees. It gradually expands the game tree by adding nodes to it. UCT considers every interior node in the tree as an independent bandit and its children (available actions at that node) as the arms of the bandit. While searching in the tree, UCT selects an action that maximizes the combination of the player's utility and a bonus similar to the one in UCB1 that is used to balance between exploration and exploitation. Implied in this selection rule is the assumptions that a player has about his opponents. When the search reaches a non-terminal leaf node, a Monte-Carlo simulation from that node to a terminal state is carried out. The value that results from the simulation is then used to update the values of all nodes along the path from the root of the tree to the node that leads to that simulation. UCT is an iterative algorithm. It searches through the tree, does simulations at non-terminal nodes and adds new nodes to the tree. Tree expansion will continue until the whole tree is expanded or a memory or time limit is reached. Proofs of convergence and regret bounds can be found in [22].

3.4 UCT Variants

The UCT algorithm was originally proposed for single-agent domains [22]. However, single agent UCT can be easily modified to be used in multiplayer games by changing the search policy in the tree. It has been applied in domains with more than one agent (*e.g.* computer Go [9, 15]) and produced notable results. It has also been used in the general game playing competition and yielded superior results for three successive years [12].

In the original version of UCT and in its simplest (and usual) implementation, UCT only considers its own utility (and not the opponents' utility) for action selection. Tree expansion is also done in a sequential manner by considering the moves of every player at a different step. We call this sequential tree expansion *sequencing*. Although it does well in a sequential two-player constant-sum game, turning a simultaneous move game to a sequential game can be troublesome. One major drawback is that when we assume that a game is sequential we are assuming information about how our opponent behaves in a game and that we know what he does in a step before we make our decision. For example, consider the simple matrix game shown in Table 3.1. The first row and column in Table 3.1 are the actions of players and the values in each cell are the payoffs for taking the joint actions crossing at that cell. The row player gets the first value while the column player gets the second one. Assume that we are playing as the row player. If we use sequencing and minimax in UCT to decide what action to take, both of our actions will be considered equally good. Because the opponent can always select an action that results in 0 points for us. However, if we do not use sequencing and consider our actions and opponent's actions simultaneously and maximize our own score, we select a_2 with no doubt. In this case if the opponent happens to be imperfect, we can gain 100 points.

	b_1	b_2
a_1	$0, 100$	$0, 100$
a_2	$100, 0$	$0, 100$

Table 3.1: The payoff matrix of a simple simultaneous move game.

The pseudocode for one iteration of our variant of the UCT algorithm is given in Algorithm 1. We use a maximization policy while searching in the game tree, which is every player is trying to maximize his own score. The policy for searching the game tree is given in lines 19-28 of Algorithm 1. In single agent case, only the value of the single player in the game is considered while traversing the tree. It should be noted that for two-player constant-sum games, the selection rule between children of a node corresponds to the minimax rule. When a game is two-player constant-sum, maximizing your own score corresponds to minimizing the opponent score at the same time and that is the way the selection rule is applied in Algorithm 1.

Each iteration of the UCT algorithm involves three phases of tree search, simulation, and value update. The first part of each UCT iteration (lines 2-10) is the tree search. As long as all the children of a node are expanded, the best child is selected to be followed in the new trajectory in this iteration. The UCT bonus is also taken into consideration for the selection of the best child to be followed (line 22). Regular UCT players just consider an exploration bonus for their own actions. However, we will consider an exploration bonus for all players. At a leaf node when there are unexpanded children, a random child is selected for expansion. This new child is also added to the tree. Regular UCT players, unlike us, usually do not consider all the children (every combination of move selection for different players) before going deeper in the game tree. Therefore, their tree expansion can be misleading based on how lucky they were in the first time that they tried a move. The second part of each UCT iteration (lines 11-16) is the simulation. If the current node is already a terminal state, then the value is just retrieved (line 12). However, if the leaf node is not a terminal state, a Monte-Carlo simulation is done from the leaf node to a terminal state to get a value for the current trajectory (lines 14-15). The third part of each UCT iteration (line 17) is the value update. The value that is gathered in the second phase is used to update the value of nodes along the trajectory in the game tree.

3.4.1 Multiplayer Games and Opponent Modeling

In general game playing the value of the goal is defined for different players in a terminal state. Thus, our program keeps values for each player instead of just keeping the value for the player who is to move in order to do more sophisticated opponent modeling. In multiplayer games, the number of children can be as large as the product of available actions for each player. Therefore, we keep the values for different actions of different players instead of keeping them for each combination of actions. This will also enable us to do more sophisticated opponent modeling than merely considering that everybody is playing against us (the paranoid assumption [36]) like what most of the

Algorithm 1 One iteration of the UCT algorithm.

```
1:  procedure UCTITERATION( )
2:      current ← root
3:      while in the tree do
4:          if all joint moves have been explored then
5:              current ← GETBESTCHILD(current)
6:          else
7:              current ← GETCORRESPONDINGCHILD(current,random-unexplored-joint-move)
8:              ADDTOTREE(current)
9:          end if
10:     end while

11:     if current is terminal then
12:         values ← GETGOALVALUES(current)    ▷ Returns the goal values for different players.
13:     else
14:         terminalNode ← MONTECARLOSIMULATION( )
15:         values ← GETGOALVALUES(terminalNode)
16:     end if

17:     UPDATEVALUES(current, values)
18: end procedure

19: procedure GETBESTCHILD(current)
20:     for every player p do
21:         for every action a of player p do
22:             values[a] ← value-of-a + C\sqrt{\frac{ln(\text{number of visits to } current)}{\text{number of times player } p \text{ has selected action } a}}
23:         end for
24:         moves[p] ← actions-of-p[\arg\max{values}]
25:     end for
26:     best ← GETCORRESPONDINGCHILD(current,moves)
27:     return best
28: end procedure

29: procedure UPDATEVALUES(current, values)
30:     while current is not root do
31:         current-selected-action-value ←                ▷ value of the selected action at current
                WEIGHTEDAVERAGE(current-selected-action-value,
                                current-selected-action-counter,values)
32:         current-selected-action-counter ← current-selected-action-counter + 1
33:         current-visits ← current-visits + 1
34:         current ← current-parent-in-this-iteration
35:     end while
36: end procedure
```

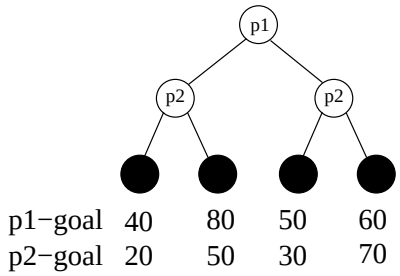


Figure 3.1: Game tree of a sample general-sum game.

general game players do.

In single agent games there is no need for opponent modeling and in two-player constant-sum games opponent modeling is limited, although work has been done in that area [7, 11]. In the single agent case, we try to maximize our own score. The situation is the same for two-player constant-sum games. By following the maximization strategy, we also achieve the minimization of the score of our opponent in two-player constant-sum games since the game is constant-sum. However, in two-player general-sum games and in multiplayer games there is no *best* modeling, which means there is no unique modeling with guaranteed *best* payoff regardless of whatever the other players do, although the paranoid assumption can be used, which guarantees a minimum playoff [36]. Therefore, if we have an incorrect model of our opponent(s), then we may suffer greatly. For example in the game tree of a sample general-sum game that is shown in Figure 3.1 if the opponent (p2) is trying to minimize our score (p1), then we are confident to get at least 50 points by selecting the right branch. However, if the opponent is trying to maximize his own score then we can get 80 points by selecting the left branch. This conflict in action selection arises from the fact that the game is general-sum.

Since we are dealing with simultaneous move games in this thesis and computing Nash equilibrium is a logical way to play these games as we will discuss later in Section 3.6.1, we use the tree search policy that gets us closest to a Nash equilibrium in a game. Nash equilibrium is a state that no player can increase his payoff by unilaterally changing his strategy, therefore assuming that every player plays to maximize his own payoff is logical to be used during tree search.

3.4.2 Exploration vs. Exploitation

The square root bonus added to the value of a node on line 22 is used to balance between exploitation and exploration. It is directly proportional to the number of times a state (parent) is visited and inversely proportional to the number of times a specific action (child) is selected. By exploiting the best action repeatedly, the bonus for selecting other actions will become larger. Therefore, exploring other actions become more favorable. The constant factor, C, defines how important it is to explore instead of selecting the greedy action. The higher the value of C, the more exploration will be done.

Although we can change the value of C to tune exploration versus exploitation, we should also take the range of the value of outcomes into consideration. If the value of C is not comparable with

C	1	40	100
1	NA	$12, 88$	$21, 79$
40	$93.5, 6.5$	NA	$67, 33$
100	$91.50, 8.5$	$34, 66$	NA

Table 3.2: Comparing the usage of variable values of C in Connect4.

the range of values, then exploration can be over emphasized or under emphasized. For example setting $C = 10000$ or $C = 1$ while the range of values is comparable with 100 will result in too much or too little emphasis on exploration. Therefore, although setting $C = 40$ can be logical in GGP while the range of values is usually from 0 to 100, if all the outcome values in a game are in the 0 to 10 interval, then a smaller value for C can be more suitable. Thus a variable value of C, dependent on the game and the game context, seems to be more desirable. A comparison between different setting of values for C is made in Table 3.2 in Connect4. The values in Table 3.2 are averaged over 100 games with start-clock equal to 30 seconds and play-clock equal to 10 seconds and memory limit of 2 gigabytes. Values in Connect4 range from 0 to 100 and $C = 40$ is the best value as discussed earlier.

3.4.3 Playout Policies

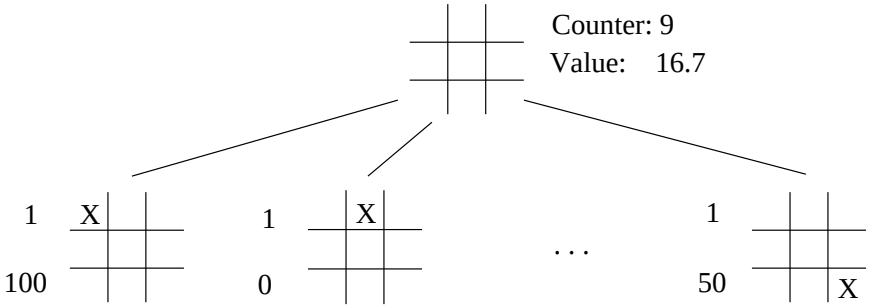
Different policies can be used to do the Monte-Carlo simulation. The simplest one is to select random actions for each player at each step. However, one can use a more informed approach by selecting the best action if the simulation runs into a node that has been visited before during the search (the idea of using transposition table [29]). In addition, history heuristic about actions can be used in the playout as well [29].

3.4.4 Updating Values

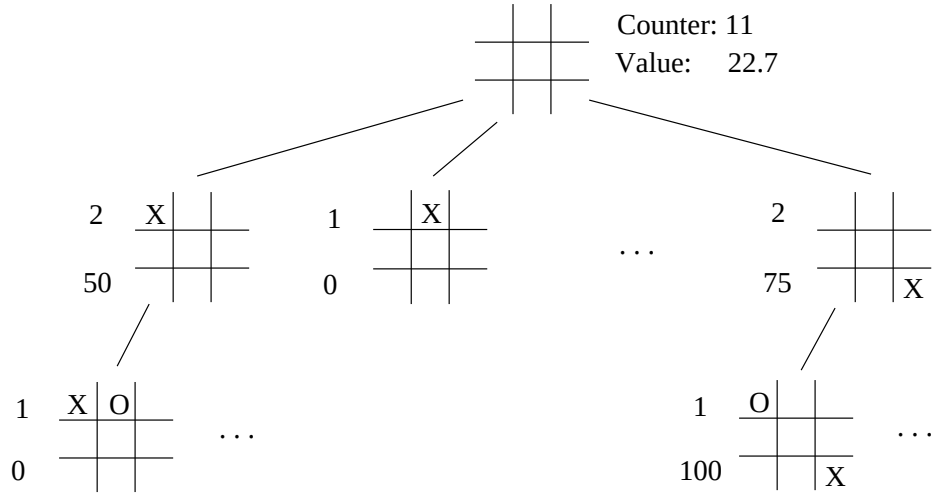
The outcome that results from a simulation will be used to update the values of the nodes along the path from the root in the search tree (line 17). Updates can simply be a weighted average. However, if the game is single-player and there is no uncertainty in the game, maximization between the old value and the new one can be used as the update rule, since in the single player case, the only concern is to achieve the highest possible outcome. In addition, a discount factor can be used during the update process to favor shorter solutions over longer ones.

3.5 UCT Example

We demonstrate how UCT expands and explores the game tree for tic-tac-toe. The rules of the game are the same as regular tic-tac-toe and the first player who gets a line on the board is the winner. The winner gets 100 points while the loser gets 0 points. A draw result is 50 points for each player.



(a) Game tree after expanding all the children of the root.



(b) Expanding game tree based on values and UCT bonus.

Figure 3.2: UCT game tree expansion in tic-tac-toe.

Since this game is two-player constant-sum, we will only consider the values for the first player.[1]

During the first nine iterations of UCT, since all the children of the root have not been expanded yet, a new child of the root is added to the game tree at each iteration. A Monte-Carlo simulation is carried out for each of them to get a value. Suppose after nine iterations the game tree and the values of nodes are analogous to what is shown in Figure 3.2(a). Without loss of generality and for simplicity, assume that the children that are not shown in Figure 3.2(a) have taken the value of zero. In the tenth iteration, the best child is selected for expansion. Since the UCT bonus is equal for all the children at this point, the child with the highest value, which is the left-most one is selected for expansion. Therefore, one of its children is added to the tree and a simulation is performed to get a value. We assume that the simulation from the newly added node results in a loss and thus the value of its parent is decreased to 50. In the eleventh iteration, the root node has two children with the value of 50. However, the UCT bonus for the right-most child is bigger than the bonus for the

[1] It should be noted that in this game, no two different joint moves will result in the same next state. Therefore, the number of visits to a child can be considered as the number of times that a particular action has been selected.

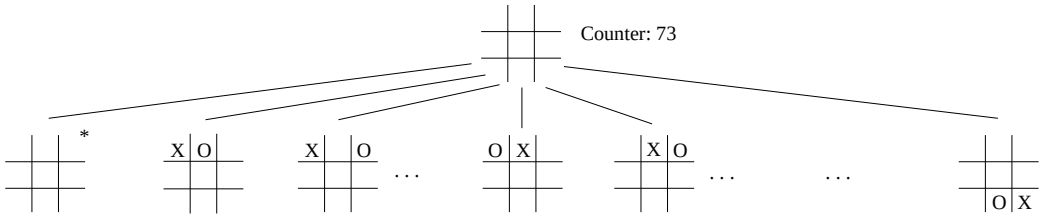


Figure 3.3: Resulted UCT game tree for simultaneous tic-tac-toe after exploring all the joint moves at the root of the tree.

left-most one as it has received less exploration. Therefore, the right-most child is selected to be followed in the trajectory in the eleventh iteration. The game tree after this iteration is shown in Figure 3.2(b).

UCT iterations continue as long as there is time left. New nodes are also added to the tree as long as there is sufficient memory. When a memory limit is reached, no new nodes are added to the tree and only the values of the nodes currently in the tree are updated. After the first move is taken in the game, the game tree above it that cannot be reached any more will be removed. This frees memory for new tree expansion and addition of new nodes to the tree. These steps will be repeated while the game is being played. When time is up for move selection at each step, the child with the highest value is selected as the move to be played.

3.6 UCT in Simultaneous Games

The pseudocode that was presented in Algorithm 1 can be used for simultaneous move games. The only difference is that in turn-taking games, only one player has a chance to select his move, while the others select the *no-operation* move. In simultaneous move games, all the players have meaningful moves to take that can change the state of the game. In addition, each game state in simultaneous move games can be the result of more than one joint move, because joint moves depend on all players and different complementary joint moves can result in a same game state.

To clarify the usage of UCT in simultaneous move games we will consider its usage in simultaneous tic-tac-toe. The rules and outcomes of simultaneous tic-tac-toe are the same as the regular one in Section 3.5. However, both players mark the board simultaneously and, if both of them mark the same cell, then the cell will remain blank. In addition, if both players get a line on the board, then it counts as a draw.

The tree expansion part is the same as in regular turn-taking games. The resulting game tree after all the joint moves at the root of the tree have been explored is shown in Figure 3.3. Although we have $9 \times 9 = 81$ different joint moves at the root, there are only $9 \times 8 + 1 = 73$ different children because of the contraction of identical game states to a single node in the tree. All the joint identical moves (those where both players mark the same cell) result in the left-most child marked with a star in Figure 3.3. After all the different joint moves have been explored, tree expansion continues

	rock	paper	scissors
rock	$50, 50$	$25, 75$	$100, 0$
paper	$75, 25$	$50, 50$	$45, 55$
scissors	$0, 100$	$55, 45$	$50, 50$

Table 3.3: Biased rock-paper-scissors payoff matrix.

deeper in the game tree and the trajectories are selected based on the selection rule (lines 19-28 in Algorithm 1).

3.6.1 Balanced Situation

When we have no model of our opponent and no *good* strategy to play a game, which is usually the case in GGP, playing according to a non-losing strategy is rational if such a strategy exists. A non-losing strategy will not lead us to a loss if we follow it while playing the game, assuming that the game is not a forced loss. It would also be better if the opponent cannot gain any advantage by unilaterally changing his strategy against us. If our strategy has all of these properties then we are playing according to a Nash equilibrium strategy. Therefore it is convenient to find and follow a Nash equilibrium in a game. Unfortunately UCT does not converge to a Nash equilibrium in general and the situation that it converges to can be easily exploited if it is known beforehand. We now give an example that UCT does not converges to a Nash equilibrium. This example serves as a counter example that UCT does not converge to a Nash equilibrium in general. How UCT can be exploited will be shown in Chapter 5.

Rock-paper-scissors with biased payoff as shown in Table 3.3 is an example of a game that UCT gets into a balanced non-Nash equilibrium situation instead of converging to the true mixed strategy Nash equilibrium.[2] The rules of the game are the same as a regular rock-paper-scissors while the outcomes are biased according to how you win the game. There is only one Nash equilibrium in this game which is a mixed strategy with action probabilities as follows.

$$P(rock) = 0.0625 \qquad P(paper) = 0.6250 \qquad P(scissors) = 0.3125$$

The expected value of Nash equilibrium for this game is 50. One possible trace of UCT with $C = 40$ will be as follows.

$(p, r), (r, p), (p, p), (s, r), (s, s), (r, s), (p, s), (s, p), (r, r), (r, r), (p, p), (r, r), (p, p), (r, r), (p, p),$
$(r, r), (p, p), (r, r), (p, p), (r, r), (p, p),$ *(r,r),(p,p),(s,s),(r,r),(p,p),(s,s),* ...

At first 9 moves, UCT explores all the different combinations of move selection for both players. By the end of the initial exploration, UCT uses the value of each action and its exploration bonus to select an action for each player. The expected value of each action, the number of times each action has been selected, and the values computed by UCT to define the next best move are given in Table 3.4 starting at step 9. Since the game is symmetric and during the iterations we assume that

[2] The purpose of this part is to give a counter example that UCT does not compute the Nash equilibrium in general.

step (N)	values (Q_a)			counters (C_a)			$Q_a + 40 \times \sqrt{\ln N / C_a}$			next actions
	r	p	s	r	p	s	r	p	s	
9	58.33	56.67	35.00	3	3	3	**92.56**	90.90	69.23	(r, r)
10	56.25	56.67	35.00	4	3	3	86.60	**91.71**	70.04	(p, p)
11	56.25	55.00	35.00	4	4	3	**87.22**	85.97	70.76	(r, r)
12	55.00	55.00	35.00	5	4	3	83.20	**86.53**	71.40	(p, p)
13	55.00	54.00	35.00	5	5	3	**83.65**	82.65	71.99	(r, r)
14	54.17	54.00	35.00	6	5	3	80.70	**83.06**	72.52	(p, p)
15	54.17	53.33	35.00	6	6	3	**81.04**	80.20	73.00	(r, r)
16	53.57	53.33	35.00	7	6	3	78.74	**80.52**	73.45	(p, p)
17	53.57	52.86	35.00	7	7	3	**79.02**	78.31	73.87	(r, r)
18	53.13	52.86	35.00	8	7	3	77.17	**78.56**	74.26	(p, p)
19	53.13	52.50	35.00	8	8	3	**77.40**	76.77	74.63	(r, r)
20	52.78	52.50	35.00	9	8	3	75.86	**76.98**	74.97	(p, p)
21	52.78	52.22	35.00	9	9	3	**76.04**	75.48	75.30	(r, r)
22	52.50	52.22	35.00	10	9	3	74.74	**75.66**	75.60	(p, p)
23	52.50	52.00	35.00	10	10	3	74.90	74.40	**75.89**	(s, s)
24	52.50	52.00	38.75	10	10	4	**75.05**	74.55	74.40	(r, r)
25	52.27	52.00	38.75	11	10	4	73.91	**74.69**	74.63	(p, p)
26	52.27	51.82	38.75	11	11	4	74.04	73.59	**74.85**	(s, s)

Table 3.4: Values in a sample trace of UCT with $C = 40$ in biased rock-paper-scissors and computation of next best action to be selected in the next iteration.

both players are using UCT, the values for both players will be identical. This equality of values on both sides results in both players selecting joint identical move, *e.g.* (r, r) which represents both players playing rock. Action selection will be done based on the values computed by UCT, which is the sum of expected value of an action and its UCT bonus (fourth column in Table 3.4). The maximum value in fourth column, which is the action to be selected next, is shown in boldface font. At step 9, rock has the highest value among all the actions and the UCT bonus is equal for all the actions. Therefore, (r, r) is selected. This selection results in paper having a higher UCT bonus that compensates for its lower value and leads to its selection. Action selection between rock and paper is switched until their expected values are lowered[3] enough that the UCT bonus for scissors can compensate its selection (step 23). From step 21, which is specified by double lines in Table 3.4, UCT gets into a cycle that does not exist.

After this sequence, the values (sum of the goal value and UCT bonus for each action) for both players in UCT will be identical which will result in both players playing the same during UCT iterations and cycling through a balanced situation. This means that players will select joint identical moves. Since the value that the players get is equal to the expected value of the Nash equilibrium that player can get, both players will be *satisfied* and not willing to deviate from their strategy. In addition, players will not get out of the balanced situation cycle, because the exploration in the UCT algorithm is deterministic. Exploration is deterministic due to the fact that it is governed by the UCT bonus and because both players play the same, the UCT bonus for different actions of each player

[3]The result of playing joint identical moves is 50 which is less than the expected values of rock and paper at this point.

	rock	paper	scissors
rock	$50, 50$	$0, 100$	$100, 0$
paper	$100, 0$	$50, 50$	$0, 100$
scissors	$0, 100$	$100, 0$	$50, 50$

Table 3.5: Regular rock-paper-scissors payoff matrix.

will be the same.

If we consider action selection ratios as the probability of choosing each action, we will get equal probabilities for each of the different actions as $(1/3, 1/3, 1/3)$ (the first, second, and third arities are probabilities of selecting rock, paper, and scissors respectively).[4] It is clear that this probability setting is not a Nash equilibrium, because either player can increase his payoff by unilaterally skewing his action selection probability toward paper.

The balanced situation that the players arrive at is dependent on the way a player models the other player. The value of C for each player can be considered as the simplest notion of opponent modeling in the UCT algorithm. For example if we consider $C = 100$ for the first player and $C = 50$ for the second player, then the probability settings for the first and second players after approximately one million iterations will be $(0.07, 0.12, 0.81)$ and $(0.03, 0.46, 0.51)$ respectively. Therefore if UCT plays as the first player, the second player can exploit UCT by skewing his action selection probability toward rock. On the whole, the balanced situation that UCT converges to is not necessarily a Nash equilibrium and can be exploited.

In addition, in the situations where UCT converges to the Nash equilibrium, *e.g.* regular rock-paper-scissors[5] whose payoff matrix is shown in Table 3.5, the way it converges to the Nash equilibrium can just be a result of luck. For example, looking into the logs of the matches of the regular rock-paper-scissors reveals that there are cases that UCT gets stuck in selecting joint identical moves, *e.g.* (r, r). Because UCT gets 50 points simply by playing joint identical moves, which is equal to the expected value of the Nash equilibrium, it will not change the way it plays. Therefore in the best case (after running for sufficient time), UCT samples all the joint identical moves equally and comes up with $(1/3, 1/3, 1/3)$ probability setting for the Nash equilibrium. It samples all the moves equally because of the UCT bonus.

It must be mentioned that UCT exploration is not random. It is a deterministic process controlled by the UCT bonus at each step. Therefore, when UCT gets into a cycle that the values of UCT bonuses for different actions repeat themselves, UCT cannot get out of the cycle. One way to solve the problem is to change the exploration in a way that will result in actually exploring every possible situation sufficient number of times. However, we will consider another algorithm, called CFR, that actually computes a Nash equilibrium.

[4] The resulted probability setting does not suggest that UCT selects its action using that distribution. Because the probability setting is generated considering the UCT bonuses while the final action will be selected solely based on the values of different actions.

[5] The only Nash equilibrium in the regular rock-paper-scissors is $(1/3, 1/3, 1/3)$ whose expected value is 50.

3.7 Conclusion

In this chapter we analyzed the UCT algorithm. We argued that it is not the best algorithm to be used everywhere, *e.g.* simultaneous move games, since it does not compute the right value everywhere, *e.g.* Nash equilibrium in case of simultaneous move games. In addition we showed that when it converges to the Nash equilibrium in a game, it can be just out of luck and cannot be relied.

Chapter 4

Computing Nash Equilibrium: the CFR Algorithm

4.1 Introduction

In this chapter we will describe the CFR algorithm and consider its use in simultaneous move games in GGP. At first we give a brief overview of the CFR algorithm (Section 4.2). After explaining immediate counterfactual regret, which is the cornerstone of the CFR algorithm, in Section 4.3, we will describe the algorithm in Section 4.4. We give examples on how to use CFR in Section 4.5. Finally we will consider how the CFR algorithm can be used in GGP (Section 4.6).

4.2 Overview

As discussed in Section 3.6.1, playing according to a Nash equilibrium in an unknown simultaneous move game is a reasonable strategy. However, if the game is complex (*e.g.* the state space is large) then we cannot compute a precise equilibrium. Instead, we can use an ϵ-Nash equilibrium strategy, where ϵ is an indication of how far we are from an equilibrium. Since we will not lose anything by following a Nash equilibrium strategy, ϵ can be considered as the amount that we will lose if we happen to play against a best response to our strategy. In fact ϵ is a measure of how exploitable we will be by following an ϵ-Nash equilibrium strategy [38].

CFR (CounterFactual Regret) is an algorithm for finding an ϵ-Nash equilibrium. It is currently the most efficient algorithm known and it can handle the largest state spaces in comparison to other available methods [38]. It also has the nice property of being incremental, meaning that the longer it runs the closer it gets to the Nash equilibrium in the environment which it is dealing with.

4.3 Immediate Counterfactual Regret

Regret is related to the difference between the maximum reward a player could have obtained versus what he did obtain. Counterfactual regret is a measure that defines how much a player regrets not

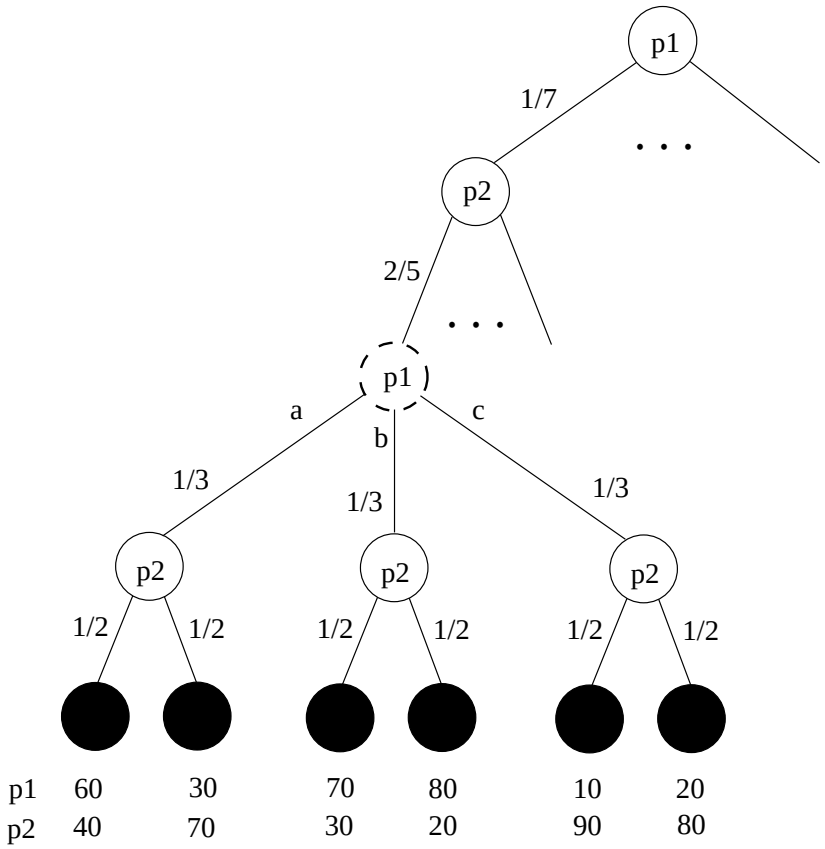


Figure 4.1: A partial game tree.

taking an action (and that is why it is called counterfactual). In the computation of the counterfactual regret for a player in a specific point in a game, it is assumed that the player himself has played to reach that point of decision making while all the other players have played according to their strategy profile.[1]

Immediate counterfactual regret of a player after a number of iterations is also defined analogously. It is the average of the player's maximum counterfactual regret over all the iterations that it is computed for. A formal definition of counterfactual regret can be found in [38]. We will demonstrate how to compute counterfactual regret with an example.

Let us compute the counterfactual regret at the node which is represented by the dashed circle in the partial game tree shown in Figure 4.1. Each node is marked with the player who has to make a decision at that point. The probability of taking each action according to the current strategy of the player is given beside each edge. Dots in the upper part of the tree mean that there are other choices that have not been shown. Dark filled nodes at the bottom of the tree are terminal nodes and the goal value for each player is given below these nodes. It should be noted that the game is constant-sum. If we compute the expected value for each of the different actions available to player

[1] A strategy profile for each player defines the probability of the player taking each action at each step of the game.

$p1$ at our desired node we have

$$E(a) = \frac{1}{2}\times 60 + \frac{1}{2}\times 30 = 45 \quad , \quad E(b) = \frac{1}{2}\times 70 + \frac{1}{2}\times 80 = 75 \quad , \quad E(c) = \frac{1}{2}\times 10 + \frac{1}{2}\times 20 = 15$$

According to current strategy probability setting of $p1$, the expected value of our desired node is $\frac{1}{3} \times 45 + \frac{1}{3} \times 75 + \frac{1}{3} \times 15 = 45$. The counterfactual regret of an action in a state is the difference between the expected value of that action in that state and the expected value of the state where the player can choose that action. Counterfactual regret of an action in particular state for a player is also weighted by the probability of other players reaching that state if they happen to play according to their strategy at that time. Therefore, $p1$'s counterfactual regret of each action in our desired node will be as follows.

$$cfr(a) = \frac{2}{5}\times (45-45) = 0 \quad , \quad cfr(b) = \frac{2}{5}\times (75-45) = 12 \quad , \quad cfr(c) = \frac{2}{5}\times (15-45) = -12$$

$\frac{2}{5}$ is the probability of $p2$ getting to the desired node. Therefore, $p1$ regrets not taking action b more than the other two actions. We explain the CFR algorithm and show how counterfactual regrets that were computed here can be used to compute the final strategy profile to play a game.

4.4 The CounterFactual Regret Algorithm

Zinkevich *et al.* state three theorems in [38] that relate immediate counterfactual regret and ϵ-Nash equilibrium. Those theorems are as follows and are the theoretical foundations of the CFR algorithm.

Theorem 1 *In a two-player constant-sum game at a specific time step, if both players' average overall regret is less than ϵ, then their average strategies are a 2ϵ equilibrium.*

Theorem 2 *Average overall regret is bounded by the sum of the independent immediate counterfactual regret values.*

Therefore, if we minimize the immediate counterfactual regret at each node the average overall regret will be minimized. To minimize the immediate counterfactual regret we just need to minimize the counterfactual regret for every action of a player at each time step, since immediate counterfactual regret is the average of the player's maximum counterfactual regret over all the iterations that it has been computed for. One way for counter factual minimization is to set the probabilities for the next iteration based on the regrets in the current iteration. Thus the probability of selecting an action with positive regret can be set relative to the ratio of its regret value to the sum of the positive regrets of all actions. If there is no action with positive regret then a uniform probability distribution over all actions can be used. If we set the probabilities in this way then our average overall regret will be decreased according to Theorem 3.

Theorem 3 *If a player selects actions based on the probability setting that is set to minimize the player's regret at each step (i.e. as just discussed) then the player's regret will be decreased relative to the square root of the number of iterations.*

The pseudocode for one iteration of the CFR algorithm is given in Algorithm 2. Proofs of convergence and bounds on how close it will get to a Nash equilibrium can be found in [38].

Algorithm 2 One iteration of the CFR algorithm.

```
1:  procedure CFRITERATION( )
2:      if root = NULL then
3:          BUILDTREE( )
4:      end if

5:      COMPUTEEXPECTEDVALUES(root)
6:      RESETALLREACHINGPROBABILITIESTOZERO( )
7:      for each player p do
8:          root.playersOwnReachingProbaility[p] ← 1
9:          root.reachingProbability[p] ← 1
10:     end for
11:     COMPUTEREACHINGPROBABILITIES(root)
12:     UPDATEPROBABILITIES(root)
13: end procedure
```

At each iteration of the algorithm, CFR computes the expected value for different actions of every player at each node (lines 10-16 in Algorithm 3). The overall expected value for each player is computed as well (lines 17-19 in Algorithm 3). It also computes the reaching probability to each node in the tree for different players. However, as CFR deals with counterfactual regret, the probability for each player is computed as if that player played to reach that node while other players have played based on their current strategy (lines 3 in Algorithm 4). In addition, it keeps track of the probability of reaching a specific node, based on the player's own strategy, to compute the final probability for action selection (line 4 in Algorithm 4). Counterfactual regrets are computed using the reaching probabilities and the difference between expected values for taking a specific action versus following the current strategy (lines 8-13 in Algorithm 5). CFR keeps track of cumulative counterfactual regret for every action of every player at each node of the tree. Action probabilities for the next iteration are computed based on the cumulative counterfactual regret. The probabilities of all the actions which have negative regrets will be set to zero as the player is suffering by taking those actions based on the current probability settings (line 19 in Algorithm 5). The probabilities of the actions which have positive regrets will be set according to the value that the player regrets them (line 17 in Algorithm 5). However, if there is no action with positive regret, then the player will switch to randomization between all of his actions using a uniform distribution (line 24 in Algorithm 5).

It should be noticed that the game is not actually being played during the computation, but the algorithm is tuning probabilities for the players to minimize their immediate counterfactual regret. While the algorithm is tuning the probabilities, it also gathers information to compute the final probabilities (lines 3-6 in Algorithm 5). The average strategy over all the iterations will be considered as the final probabilities for selecting each action while a player is actually playing the game. The

Algorithm 3 Computing expected values in CFR.

```
1:  procedure ComputeExpectedValues(root)
2:      for each child c of root do
3:          ComputeExpectedValues(c)
4:      end for

5:      for each player p do
6:          for each a ∈ actions(p) do
7:              root.actionExpectedValue[p][a] ← 0
8:          end for
9:      end for

10:     for each child c of root do
11:         for each player p do
12:             pAct ← c.action[p]
13:             prob ← Π_{op≠p} root.actionProbability[op][c.action[op]]
14:             root.actionExpectedValue[p][pAct] += prob × c.expectedValue[p]
15:         end for
16:     end for

17:     for each player p do
18:         root.expectedValue[p] ←
                Σ_{a∈actions(p)} root.actionProbability[p][a] × root.actionExpectedValue[p][a]
19:     end for
20: end procedure
```

Algorithm 4 Computing reaching probabilities in CFR.

```
1:  procedure ComputeReachingProbabilities(root)
2:      for each child c of root do
3:          c.reachingProbability[p] +=
                root.reachingProbability[p] × Π_{op≠p} root.actionProbability[op][c.action[op]]
4:          c.playersOwnReachingProbaility[p] =
                root.playersOwnReachingProbaility[p] × root.actionProbability[p][c.action[p]]
5:          if AllParentsComputationAreDone(c) then
6:              ComputeReachingProbabilities(c)
7:          end if
8:      end for
9:  end procedure
```

Algorithm 5 Updating probabilities in CFR.

```
1:  procedure UPDATEPROBABILITIES(root)
2:      for each player p do
3:          for each a ∈ actions(p) do
4:              root.cfrActionProbability[p][a] +=
                    root.playersOwnReachingProbability[p] × root.actionProbability[p][a]
                  ▷ Keeps track of accumulative probabilities to extract probability of actions at last
5:          end for
6:          root.cfrReachingProbability[p] += root.playersOwnReachingProbability[p]
                ▷ The final probability for player p taking action a will be cfrActionProbability[p][a] / cfrReachingProbability[p]
7:          sum ← 0
8:          for each a ∈ actions(p) do
9:              root.regret[p][a] += root.reachingProbability[p] ×
                                        (root.actionExpectedValue[p][a] - root.expectedValue[p])
10:             if root.regret[p][a] > 0 then
11:                 sum += root.regret[p][a]
12:             end if
13:         end for
14:         if sum > 0 then
15:             for each a ∈ actions(p) do
16:                 if root.regret[p][a] > 0 then
17:                     root.actionProbability[p][a] ← root.regret[p][a] / sum
18:                 else
19:                     root.actionProbability[p][a] ← 0
20:                 end if
21:             end for
22:         else
23:             for each a ∈ actions(p) do
24:                 root.actionProbability[p][a] ← 1/|p's actions|
25:             end for
26:         end if
27:     end for
28: end procedure
```

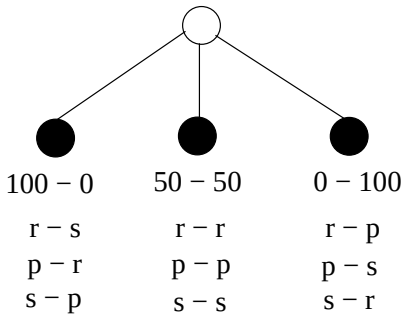


Figure 4.2: Rock-paper-scissors game tree for CFR. Dark nodes are terminal.

probability of selecting an action in the average strategy is the probability of selecting that action in each strategy during all the iterations, weighted by the probability of the strategy reaching that point of decision making.

4.5 CFR Example

4.5.1 First Example

We consider the game of rock-paper-scissors to illustrate how CFR works (refer to Figure 4.2). Assume the first player's ($p1$) action probabilities are $(1, 0, 0)$ (the first, second, and third arities represent the probability of playing rock, paper, and scissors, respectively) and the second player's ($p2$) action probabilities are $(0, 1, 0)$. Considering the probability settings, the expected value for $p1$ playing rock will be as follows.[2]

$$\begin{aligned} E_{p1}(r) &= P_{p2}(r) \times goal_{p1}(r, r) + P_{p2}(p) \times goal_{p1}(r, p) + P_{p2}(s) \times goal_{p1}(r, s) \\ &= 0 \times 50 + 1 \times 0 + 0 \times 100 \\ &= 0 \end{aligned}$$

The expected value for playing paper and scissors will be 50 and 100 respectively. Therefore the current expected value for the first player will be as follows.

$$\begin{aligned} E_{p1} &= P_{p1}(r) \times E_{p1}(r) + P_{p1}(p) \times E_{p1}(p) + P_{p1}(s) \times E_{p1}(s) \\ &= 1 \times 0 + 0 \times 50 + 0 \times 100 \\ &= 0 \end{aligned}$$

The counterfactual regret for each of the available actions to $p1$ will be as follows.

$$cfr_{p1}(r) = E_{p1}(r) - E_{p1} = 0 - 0 = 0$$

$$cfr_{p1}(p) = E_{p1}(p) - E_{p1} = 50 - 0 = 50$$

[2] $goal_{p1}$: $p1$'s goal value, r: playing rock, p: playing paper, s: playing scissors, (r, r): a state where both players play rock (the first and the second arities correspond to the $p1$ and $p2$ actions respectively).

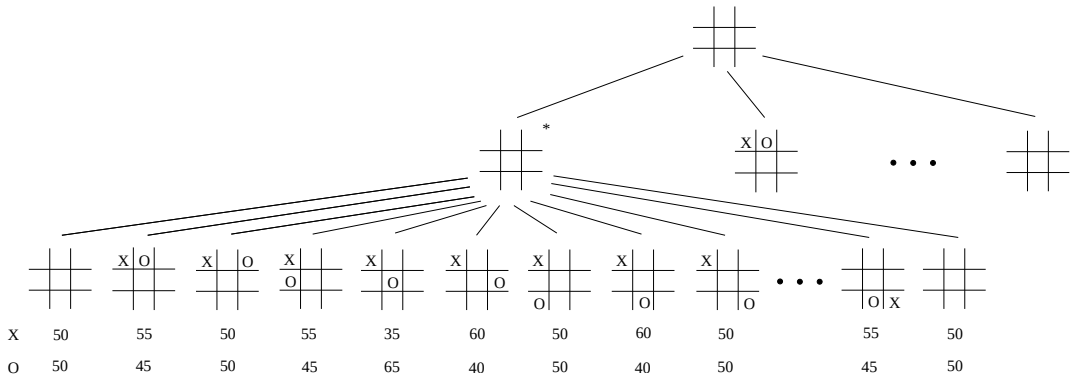


Figure 4.3: Simultaneous tic-tac-toe partial game tree for CFR.

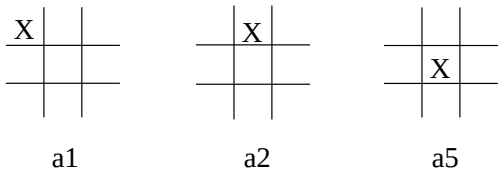


Figure 4.4: Three different moves of x-player in simultaneous tic-tac-toe.

$$cfr_{p1}(s) = E_{p1}(s) - E_{p1} = 100 - 0 = 100$$

Therefore, the action probabilities for $p1$ will be updated for the next iteration as follows.

$$(0/(100+50), 50/150, 100/150) = (0, 1/3, 2/3)$$

Similar computations will be done for the second player and his action probabilities will be updated as well before the next iteration.

4.5.2 Second Example

We consider simultaneous tic-tac-toe to illustrate how CFR works in a more complex game. The rules of the game are the same as the regular tic-tac-toe except if both players mark the same cell, the cell will remain unmarked in the next step. Wins, draws, and losses result in a reward of 100, 50, and 0 respectively. If both players manage to complete a line of their own, the game will be considered as a draw. A partial game tree of the game is shown in Figure 4.3. Expected values for each player are given beneath each leaf node. Identical states are not coalesced for simplicity. Assume both players select their actions at each step using a uniform distribution. We describe how to compute counterfactual regret and update action selection probabilities for the actions[3] shown in Figure 4.4 at the node marked with star in Figure 4.3. Let us assume the expected value for the x-player at the stared node is 50. The expected value for x-player playing $a1$ is 51.67.

$$E_x(a1) \quad = \quad P_o(a1) \times E_x(a1_x, a1_o) + P_o(a2) \times E_x(a1_x, a2_o) + \ldots + P_o(a9) \times E_x(a1_x, a9_o)$$

[3] $a1$-$a9$ correspond to marking each of the cells in the tic-tac-toe board.

$$
\begin{aligned}
&= \frac{1}{9} \times 50 + \frac{1}{9} \times 55 + \frac{1}{9} \times 50 + \frac{1}{9} \times 55 + \frac{1}{9} \times 35 + \frac{1}{9} \times 60 + \frac{1}{9} \times 50 + \frac{1}{9} \times 60 + \frac{1}{9} \times 50 \\
&\approx 51.67
\end{aligned}
$$

Let us assume that $E_x(a2) = 45$ and $E_x(a5) = 65$. Therefore, the counterfactual regret for each of the actions under consideration will be as follows.

$$cfr_x(a1) = \frac{1}{9} \times (51.67 - 50) \approx 0.19$$

$$cfr_x(a2) = \frac{1}{9} \times (45 - 50) \approx -0.56$$

$$cfr_x(a5) = \frac{1}{9} \times (65 - 50) \approx 1.67$$

Let us also assume that the sum of the positive regrets of the other actions of x-player is 0.57. Thus, the new probabilities for each of the actions under consideration will be as follows.

$$P_x(a1) = \frac{0.19}{2.43} \approx 0.08 \quad , \quad P_x(a2) = 0 \quad , \quad P_x(a3) = \frac{1.67}{2.43} \approx 0.69$$

This kind of computation will be done for other actions of x-player as well as those of the o-player and all the probabilities will be updated before starting the next iteration.

4.6 CFR in GGP

CFR was originally designed for poker which is an imperfect information game [38]. The current state of a game in an imperfect information game is based in part on hidden information. Therefore the current state of the game can only be defined to be among a set of states at any moment in the game. Each set of these states is called an information set.

Definition 5 *An **information set** for player p in a game is a collection of game states among which p cannot distinguish.*

For example at the beginning of a poker game all the states that can be built based on the player's own set of cards and different assumptions for the cards in the opponent's hand can be considered as an information set. Figure 4.5 shows a chance node and information sets in an imperfect information game. The game tree in Figure 4.5 is a partial game tree that just shows one step of the game. Each player can take 2 different actions which are represented by the edges marked with the action names. Decision making nodes are shown with circles and the player who must take an action is given in the circle. Because of the hidden information (caused by the chance node), players can end up in subtrees that are in the same information sets (same dashed shapes) and are not distinguishable.

However, the only source of imperfect information in GGP is the result of simultaneous actions taken by different players. This simplifies the use of CFR in GGP since there is no hidden information in the game except the nondeterminism of what action the opponent will select at each step. Therefore, each information set in GGP is in fact a unique state. In GGP when we reach a state, we

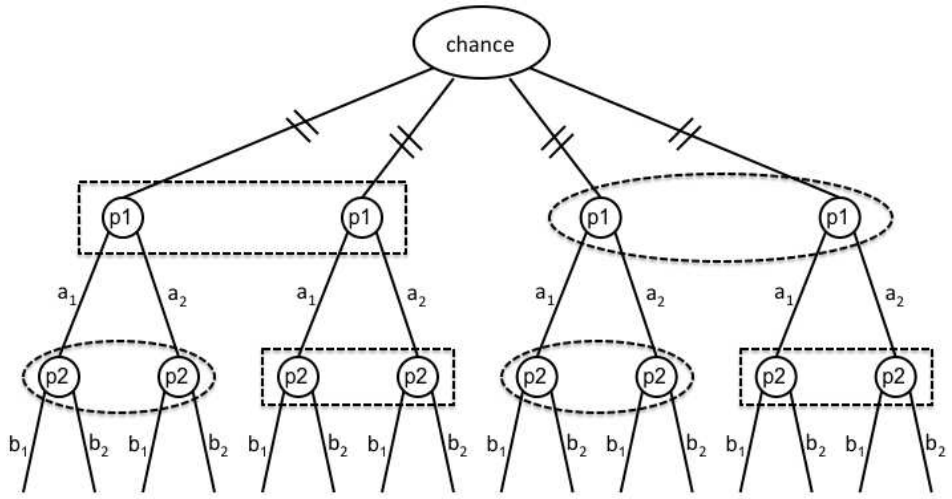
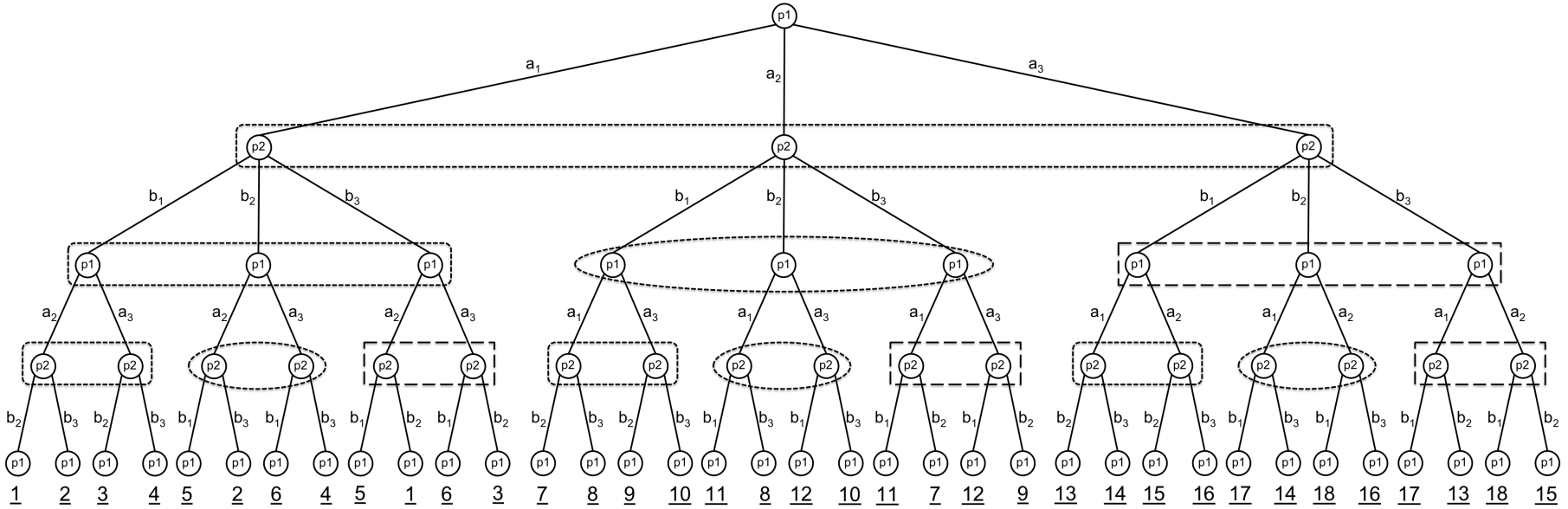


Figure 4.5: Demonstration of chance node and information sets in an imperfect information game.

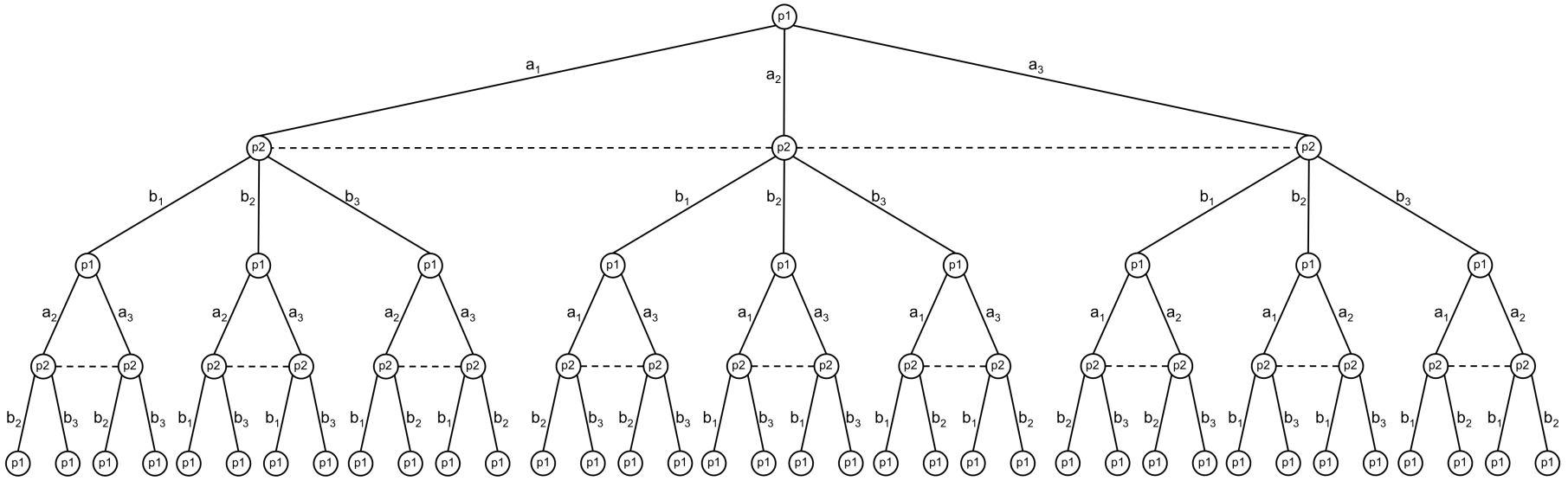
exactly know where in the game we are and we do not need a complete history of our match from the beginning of the game to find it out. Thus we can truncate parts of the tree that are not reachable anymore and resume tree expansion for new states. Therefore, the determinism of the games in GGP suits CFR very well.

An example that compares the importance of history in an imperfect information game even without any chance nodes versus a perfect information game is given in Figure 4.6. The game trees are partial game trees that just show two steps of the games and we use the same conventions as in Figure 4.5. To improve the distinction of information sets at the leaf nodes in Figure 4.6(a), the number of the information set that each node belongs to is given instead of grouping the states in the same information set using dashed shapes. Dashed lines are used in Figure 4.6(b) to show simultaneous moves and the fact that players have no knowledge about the action that the other player takes in the same step.

The partial game tree given in Figure 4.6(a) corresponds to the following game. It is a two-player simultaneous move game in which each player has 3 cards. At the beginning, each player decides to put aside one of his cards without informing the other player. Therefore, the first move of both players is hidden. Then, on each turn each player plays one card. The first player who plays the card that the other player has put aside at the beginning wins the game. If both players' guesses happen to be correct in the same step or they put aside the same cards, the game is a draw. Since there is hidden information, *e.g.* a hidden move, in an imperfect information game, after taking actions we can only say in what information set we are and not the specific state. For example, a trivial information set is the first time that $p2$ must take an action. Since he does not know about the action that $p1$ is going to take, all the three nodes at depth one of the tree are in the same information set (root is at depth zero). However more interesting information sets are where the players take their second action. Due to the fact that the first action of each player is hidden, information sets are separated based on



(a) Partial game tree sample of imperfect information game.



(b) Partial game tree sample of perfect information game.

Figure 4.6: Information set depiction in imperfect and perfect information games.

the action that the player has taken himself (depth two of the tree). This fact is true at the third time that $p1$ wants to take an action (at the leaves). All the nodes in the same information set share the same information. For example in any of the leaf nodes in information set number 1, $p1$ knows that he has hidden a_1 and played a_2 while $p2$ has not hidden b_2. Solving this game using CFR requires the whole game tree to be aware of the history of the game.

However, this game cannot be used in GGP, because no hidden move is permitted in GGP. Therefore, at the end of every step each player knows what others have done and every information set will just be a single state. The game tree for the transformation of this game into GGP context (first action is also known) is given in Figure 4.6(b). As it can be seen, every time that $p1$ must take an action, the game state can be defined as a single node. Therefore, there is no need to keep the whole tree and tree expansion can be done sequentially by removing obsolete nodes and adding new ones.

Although determinism of the games in GGP makes the application of CFR easier on one hand, but on the other hand using any abstractions while dealing with games in GGP is not as easy as using abstraction in specific games, *e.g.* poker, to shrink the state space. In poker, the game is known in advance and well designed abstractions can be used, but in GGP no good way of doing the abstraction is known. Therefore, CFR must deal with a game tree that will grow as the state space grows.

CFR expands the game tree at first by considering all the joint moves of the different players at each step of the game. However, if the whole game tree is too large to fit in memory, we only expand the tree to a certain depth. Since we need return values for different players at the leaf nodes, simulations can be done to obtain these values. A simulation involves playing a sample match from the leaf node to a terminal node. In poker since the computation is done offline, as much as time and memory that is required can be used to compute the final strategy to play the game. However, in GGP the player must submit his moves before a time limit is reached. Therefore deciding on the size of the tree that we must deal with is a critical issue. The smaller the tree is, the faster it will be to do an iteration over the tree and the values will converge faster. In case of partial trees, we will have non-terminal leaves in our tree that we need to evaluate. We must do simulations to acquire a value for our CFR computation. If we just expand a small portion of the game tree then the simulation trajectories will be longer and the outcome is more variable (compared to shorter trajectories) implying that we will need a higher number of simulations. Thus, although the probabilities that CFR computes will converge faster, they will be farther from the actual values that we must converge to. The reverse is true if we expand a larger portion of the tree. While it takes longer to converge, we will converge to a higher quality solution. Therefore, there is a trade off between how fast we can get a stable probability setting versus how good the result will be.

In addition to selecting an appropriate depth for tree expansion in CFR, all the simulations at leaf nodes can be either done at first or during each iteration for CFR when we run into a non-terminal

leaf node. While the first approach is faster, the second approach will result in better long-term quality since the quality is not bounded by the simulations that have been done at first.

In the next chapter we will present experimental results that show how CFR plays against UCT in simultaneous move games. In addition, we will consider situations where CFR has a great advantage over UCT.

4.7 Conclusion

In this chapter we described counterfactual regret and the CFR algorithm for computing ϵ-Nash equilibrium. We gave examples on how to compute counterfactual regret and how the algorithm works. We explained the usage of CFR in GGP and different trade-offs that should be considered.

Chapter 5

Experimental Results

5.1 Introduction

In this chapter we provide experimental results showing how CFR players with different settings perform against each other. Experimental results demonstrating the performance of a CFR player versus a UCT player are also provided. We give an example that using a CFR player instead of a UCT player result in great advantage as well. In addition, we discuss how following a Nash equilibrium may not be the best strategy in all situations. We will then explain how a known model can be used by CFR for exploitation while managing not to have a brittle strategy that is exploitable itself. Finally, we present experimental results demonstrating the effectiveness of our approach, but pointing to further open issues.

5.2 CFR Demonstration

5.2.1 CFR convergence rate

To give an idea about how quickly we converge to a Nash equilibrium in a game, we computed the MSE of the estimated probabilities over iterations until convergence to an equilibrium for biased rock-paper-scissors (discussed in Section 3.6.1), 9 repetition of repeated rock-paper-scissors, and Goofspiel with 5 cards.

Repeated rock-paper-scissors is a regular rock-paper-scissors repeated for specific number of times (9 times in our case). After the last repetition, the player who won more rounds gets 100 points and the loser gets 0 points. Draws result in 50 points for each player. The GDL for repeated rock-paper-scissors is given in Appendix B.

Goofspiel, also known as the *Game of Pure Strategy* (GOPS), is a card game for two or more players. The variant that we are considering here is a two player game with three suits of cards, all facing up. Therefore there is no hidden information in the game. The number of cards can be variable, *e.g.* from ace to 5 inclusive. Each player owns a suit and the third suit is on the ground in a specific order. For convenience, we assume that the third suit is in order (from ace to 5). At each step

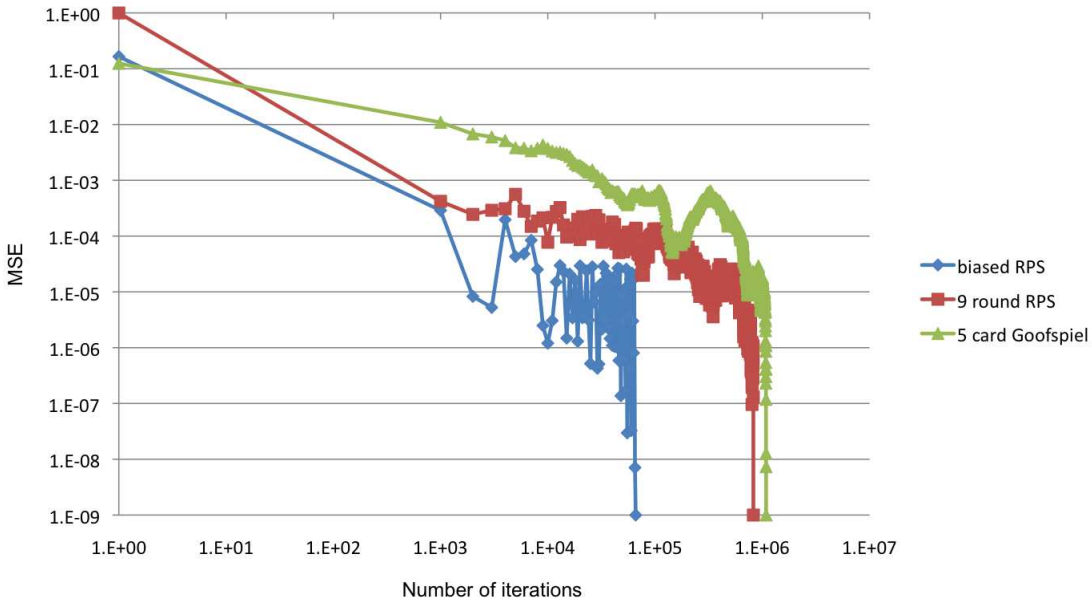


Figure 5.1: CFR convergence rate.

game	# nodes	# terminal nodes	$MSE < 0.01$		$MSE < 0.001$	
			# iterations	time (sec)	# iterations	time (sec)
biased RPS	10	9	17	≈ 0	87	≈ 0
9 round RPS	220	55	45	≈ 0	236	0.02
5 card Goofspiel	2,936	138	1,114	1.16	29,730	31.09

Table 5.1: Convergence rate of CFR in trees with different sizes.

of the game, each player selects a card in his hand and both players announce their selected cards simultaneously. The player who selects a higher card will gather the card placed on the ground from the third suit and acquire as many points as the value of the card (1 to 5 for ace to 5). Picking up cards will be done in the order they are placed on the ground (from ace to 5 in this example). For example, at the first step if one player plays ace and the other one plays 2, the player who plays 2 will get the ace from the third suit and accumulates 1 point. If both players happen to have the same card, no one will win the card from the third suit and all three cards will be discarded. At the end of the game, the player with the higher points wins 100 points and the other player gets 0. A draw results in 50 points for each player. The GDL for Goofspiel with 5 cards is given in Appendix C.

The graph in Figure 5.1 shows the result for CFR convergence rate for the three aforementioned games. Both axis are in logarithmic scale. The horizontal axis is the number of iterations and the vertical axis is the MSE of our estimation of a Nash equilibrium probabilities. Total number of nodes as well as terminal nodes in the tree of each game are given in Table 5.1. Number of iterations and time in seconds until MSE is less than 0.01 and 0.001 are also given. Tree sizes seem small (in comparison to poker), but larger tree sizes (comparable to poker) are GGP inefficient since all the computations are done online during the game.

In these experiments, we let CFR converge to an equilibrium at first. Then we record the probability setting of the equilibrium and run CFR again. In this phase, we also compute an estimation error at desired iterations. Let S be the set of all game states and $A(i,s)$ be the set of all actions that player i can take in state s. Also assume that the probability of selecting action a in state s by player i in iteration T is $P_i^T(s,a)$. We use $P_i^*(s,a)$ to refer to the final probability that player i uses to select action a in state s when CFR has converged to an equilibrium. The error that we compute is defined as follows.

$$\frac{1}{N} \sum_{i \in players} \sum_{s \in S} \sum_{a \in A(i,s)} \left(P_i^T(s,a) - P_i^*(s,a)\right)^2 \tag{5.1}$$

where

$$N = \sum_{i \in players} \sum_{s \in S} |A(i,s)| \tag{5.2}$$

We assume to be in an equilibrium if the following condition holds.

$$\max_{i \in players} \left(\max_{s \in S} \left(\max_{a \in A(i,s)} \left| P_i^T(s,a) - P_i^{T-1}(s,a) \right| \right) \right) \leq \epsilon \tag{5.3}$$

Equation 5.3 simply states that the maximum amount of change in the probability of selecting an action a at any state s by any player i in the game between two consecutive iterations $(T-1, T)$ must be less than or equal to ϵ for the situation of CFR to be considered as converged. We used $\epsilon = 10^{-6}$ in our experiments and initialized the probabilities using uniform distribution except for repeated rock-paper-scissors that uniform distribution is a Nash equilibrium. We set the probability of selecting the first action at each step, *e.g.* paper, equal to one while initializing the probabilities for rock-paper-scissors.

It can be seen that CFR converges faster in smaller games. But the interesting point is that while convergence needs a lot of iterations, *e.g.* millions of iterations in Goofspiel with 5 cards, the MSE decreases rapidly at the early stages (the MSE drops to less than 0.001 after approximately $30,000$ iterations in almost 30 seconds[1]). However, it should be mentioned that these graphs are not an indication of how the quality of the solution found by CFR improves over time. The best response must be computed for that purpose, because there can be more than a single Nash equilibrium in a game. Therefore, it can be the case that while CFR is getting closer to a Nash equilibrium, it is in fact getting farther from another Nash equilibrium. Fortunately, CFR will not just wander around and the more iterations that CFR runs, the closer it gets to a Nash equilibrium as discussed in Section 4.4. In addition, it should be noted that although we may suffer against a best response, but opponents cannot compute the best response against us. Because they do not have a model of us in advance and they cannot build one in the current setting of the competition that most games are played only once against a specific player.

[1]Comparing the number of iterations and the time that CFR requires to lower error margins in the Nash equillibrium, *viz.* MSE less than 0.01 and 0.001, also gives a good idea about the CFR convergence rate.

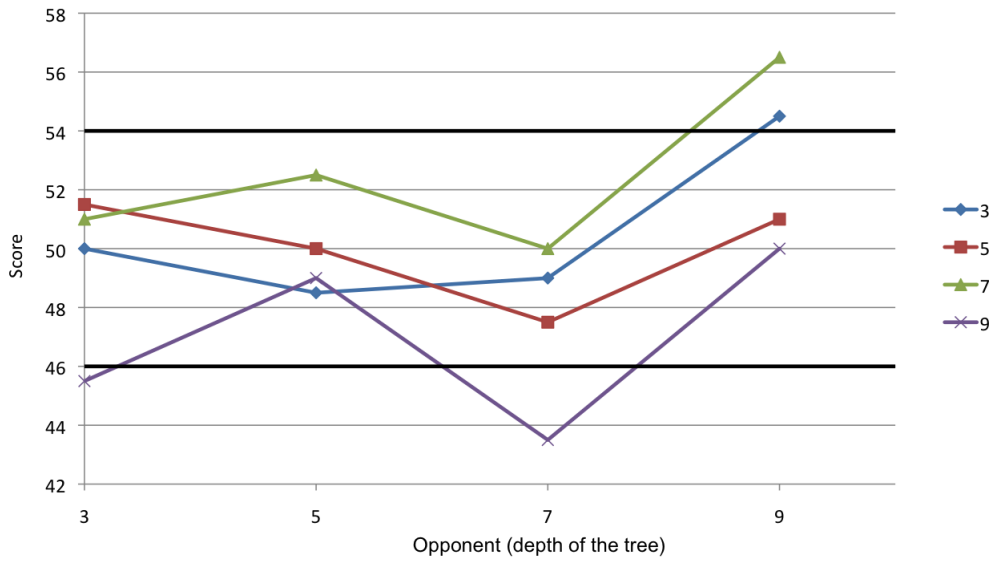


Figure 5.2: CFR players with different tree sizes playing against each other in repeated rock-paper-scissors (9 repetitions) (start-clock $= 30sec$ and play-clock $= 10sec$).

5.2.2 The Effect of Tree Size in CFR performance

To demonstrate how the size of the tree can effect the quality of a CFR player in GGP, we compare how CFR players with different limits on tree sizes (game trees that are expanded to different depths) play against each other. The policy that we used to get a value at non-terminal leaf nodes during CFR computations was to run 1000 simulation for each leaf in batches of 25 simulations. We stop simulations when the difference between the running average of values from one batch to the next was smaller than 1 point. The results for such tournaments for 9 repetitions of repeated rock-paper-scissors and Goofspiel with 5 cards are given in Figures 5.2 and 5.3. We initialized the probabilities in Goofspiel with 5 cards using uniform distribution. But we set the probability of selecting the first action at each step, *e.g.* paper, equal to one while initializing the probabilities for rock-paper-scissors since the uniform distribution is itself a Nash equilibrium in this game.

In both graphs (Figures 5.2 and 5.3) each line represents a player with a specific depth limit on the tree expansion. The legends that describe which line corresponds to what depth of tree expansion are given on the right hand side of the graphs. In both graphs, the horizontal axis indicates the depth of tree expansion being used by the opponent and the vertical axis indicates the score. Therefore, each point in the graphs corresponds to a match between two restricted depth expansion CFR players (one corresponding to the depth of the line representing a specific depth and the other one corresponding to the player in the horizontal axis). The values in the vertical axis corresponding to each point represent the average score that the line player obtained over 100 games. Both games are constant-sum and the values sum up to 100. Values greater than 50 and less than 50 can be considered as wins and losses respectively. Therefore, the higher the line is, the better the player corresponding to that

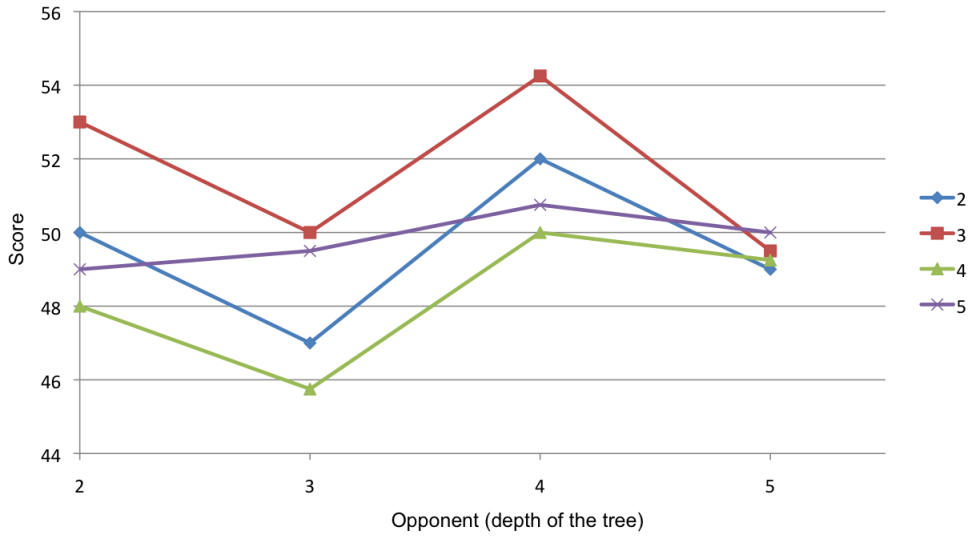


Figure 5.3: CFR players with different tree sizes playing against each other in Goofspiel with 5 cards (start-clock $= 30sec$ and play-clock $= 10sec$).

line is (or the lower the values in a column is, the better the player corresponding to that position in the horizontal axis is). Based on the graphs the player with depth 7 in rock-paper-scissors and depth 3 in Goofspiel are the best players among their groups. In Section 4.6 we discussed about trade-offs in depth selection and depths 7 and 3 are the breakpoints in these two games. However, it should be mentioned that the differences less than 4 points are only 50 percent statistically significant at most. Greater differences are more than 85 percent statistically significant using the t-test. Since many of the differences are small, it can be considered that the simulation policy used at leaf nodes were effective as well. In addition, the fact that the games are not adversarial, *e.g.* the advantages in a game does not turn into disadvantages because of a specific setting at a particular state, is well suited for the sampling approach at the leaf nodes in partial trees.

It is intuitive that the larger the tree is the better the quality of the player must be. However, according to the results in the graphs in Figures 5.2 and 5.3, it is better not to expand the whole tree when there is not enough time to run sufficient number of iterations to converge to an *acceptable* probability setting. Therefore, if we increase the time, the quality of the players with deeper trees must improve. To test this hypothesis, we ran the same experiments with longer start-clock. The results are presented in Figures 5.4 and 5.5. As it can be seen, in both games the quality of the players with deeper trees improved.

The lines that separate the statistically significant differences (more than 54 or less than 46 for repeated rock-paper-scissors) are shown in Figures 5.2 and 5.4. If we compare the graphs, the player with the whole tree (depth 9) loses statistically significant against players with smaller trees in the former, while he improves his performance given the longer stat-clock in the latter. Using a 60 second start-clock, the player with the whole tree only loses against one player with partial tree

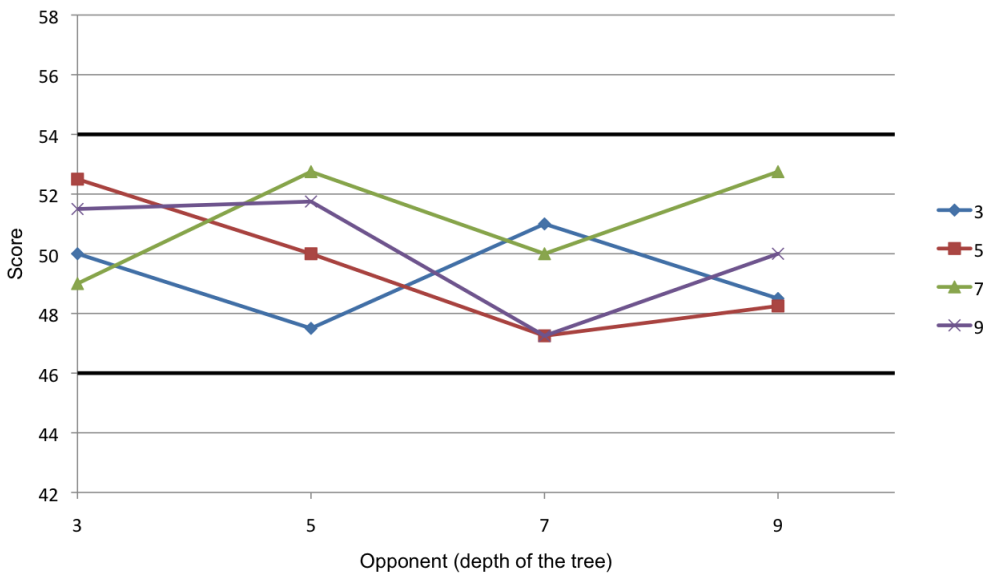


Figure 5.4: CFR players with different tree sizes playing against each other in repeated rock-paper-scissors (9 repetitions) (start-clock $= 60sec$ and play-clock $= 10sec$).

	start-clock (sec)	play-clock (sec)	CFR	UCT
repeated rock-paper-scissors (9 repetition)	30	10	52	48
simultaneous breakthrough (3x5)	30	10	48.25	51.75
simultaneous tic-tac-toe	30	10	50	50
Goofspiel (5 cards)	30	10	55	45
Goofspiel (7 cards)	60	10	53.25	46.75

Table 5.2: Comparing the usage of CFR and UCT in several simultaneous move games.

(depth 7), which is not statistically significant anymore. Similarly, comparing Figure 5.3 and 5.5 shows improvement in players with deeper trees in Goofspiel with 5 cards. It can be seen that the deeper the tree is expanded, the better the performance of the player become for longer start-clock.

5.3 Comparing CFR Against UCT

To demonstrate how a CFR player performs against a UCT player in a simultaneous move game, we performed experiments with repeated rock-paper-scissors, simultaneous breakthrough, simultaneous tic-tac-toe, and Goofspiel with 5 and 7 cards.

Breakthrough is a game played on a regular chess board. However, each player has two rows of pawns instead of regular chess pieces as shown in Figure 5.6. Pawns can move ahead one square either straight or diagonally, but captures must be done diagonally. The first player who manages to get one of his pawns to the last row of the opponent wins the game.

Simultaneous breakthrough is a variant of the breakthrough game in which both players move simultaneously. If both players move to the same cell simultaneously, the conflict will be resolved by considering the *privilege* of the players. The player who has the privilege will get his piece settled

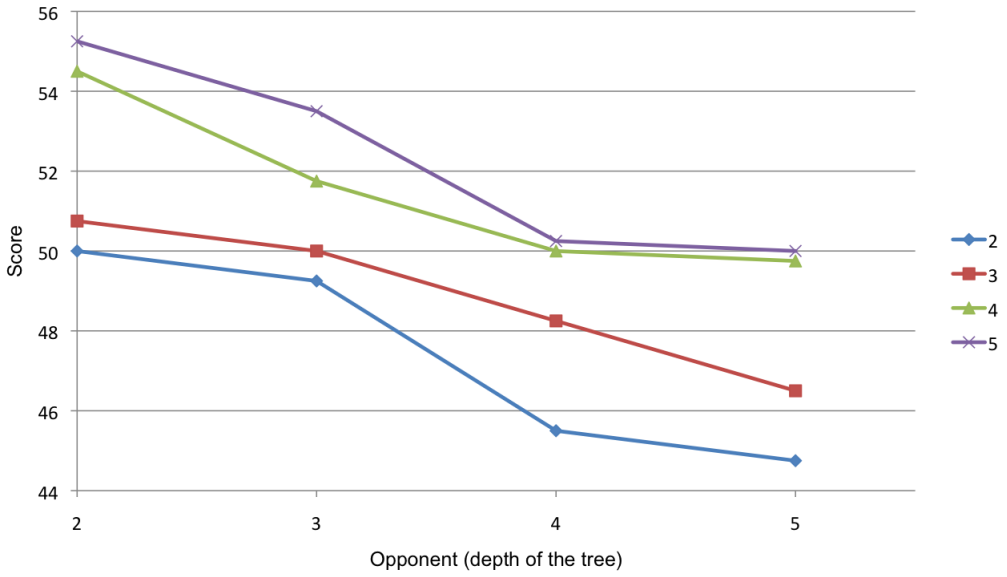


Figure 5.5: CFR players with different tree sizes playing against each other in Goofspiel with 5 cards (start-clock $= 60sec$ and play-clock $= 10sec$).

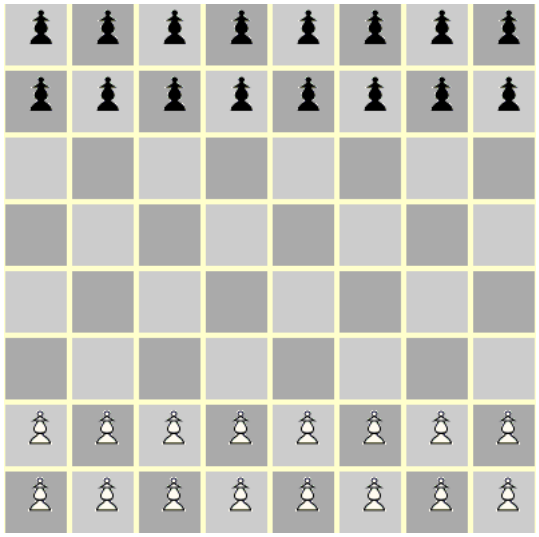


Figure 5.6: Breakthrough inital state.

in the cell. At the first state of the game, the first player has the privilege. The privilege switches between players after each collision. In addition, if both players manage to get to the last row of the opponent, the game will be considered as a draw. The GDL for simultaneous breakthrough is given in Appendix D. Since the way that we changed the game to a simultaneous move game results in the state space growth, we used smaller board size that is easier to handle, *e.g.* 3×5.

Simultaneous tic-tac-toe is a simultaneous variant of regular tic-tac-toe. However, if both players mark the same cell, the cell will remain unmarked in the next step. The game ends after 9 steps. The player who manages to form a line gets 100 points and the loser gets 0 points. If no line is formed or both players form a line of their own, the game counts as draw resulting in 50 points for each player.

Table 5.2 shows the results that are averaged over 200 games, with each player playing 100 times on each side in case the outcome of a game is biased toward a specific seating. The length of the start-clock and play-clock is also given in the table. In these experiments an implementation of UCT (as described in Chapter 3) is used versus an implementation of CFR (as discussed in Chapter 4). The UCT player is an enhanced version of our general game player that participated in the 2008 GGP competition. Uniform distribution is used to initialize the initial probabilities in CFR except for repeated rock-paper-scissors that uniform distribution is a Nash equilibrium. We used longer start-clock for Goofspiel with 7 cards in order to let CFR build the whole game tree.

The best result for CFR is in Goofspiel with 5 cards where there is a 10 point difference in the scores. This is more than 95 percent statistically significant using the t-test. However the difference in none of the other games is statistically significant. Therefore the only conclusion that can be drawn from these set of experiments is that given enough amount of time, CFR can compete with UCT. In addition, CFR performing slightly better in Goofspiel with 7 cards where the start-clock was just enough to build the game tree, is an indication of the importance of initial values in CFR. Uniform initialization seems to payoff in Goofspiel with 7 cards.

To compare how the length of start-clock and play-clock can effect the quality of UCT and CFR, we ran experiments with different settings of start-clock and play-clock in Goofspiel with 5 cards. The results of such experiments are given in the graphs in Figures 5.8 and 5.7. In both graphs, the vertical axis is the score of the player averaged over 200 games. The horizontal axis is play-clock in Figure 5.8 and start-clock in Figure 5.7. 95 percent confidence interval around each data point is also shown. As it can be seen in the graphs, neither CFR nor UCT has a clear advantage over the other one for different settings of play and start clocks. However it can be said that the performance of both algorithms improves given more time. It should also be stated that the probabilities in CFR are initialized using uniform distribution in these experiments and when the time is too short to do any iterations, uniform distribution is actually used for action selection. Not surprisingly, UCT does not perform well given very short time and therefore both algorithms can compete again. Thus performing slightly better for very short times does not necessarily imply that CFR has an advantage, but is an indication of the effect of good initialization.

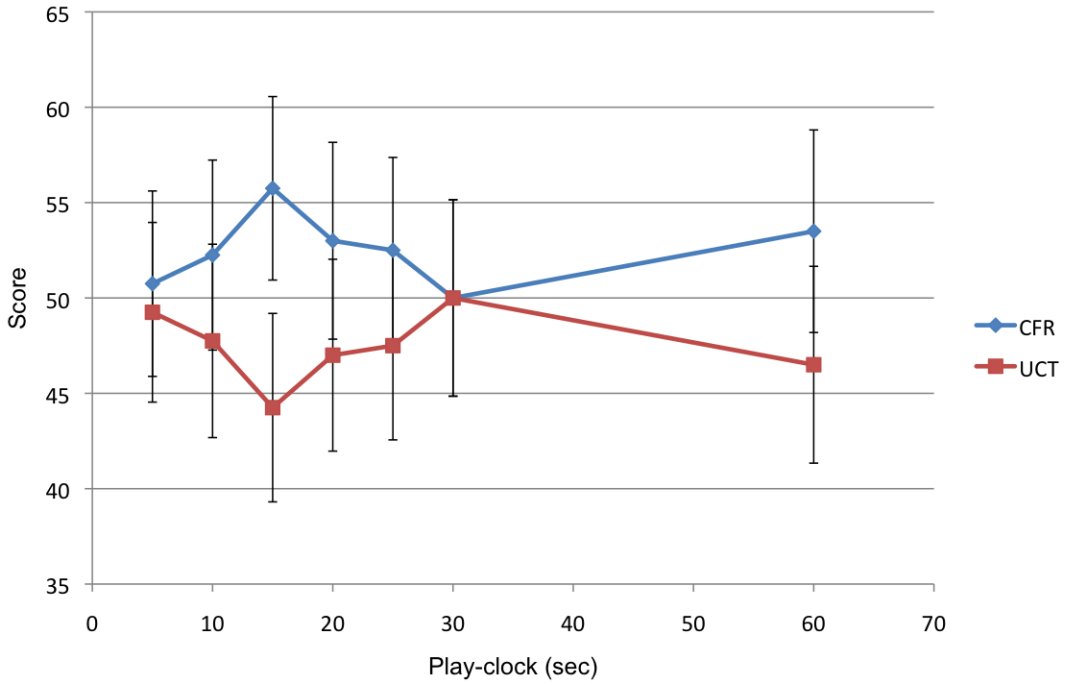


Figure 5.7: Comparing CFR and UCT in Goofspiel with 5 cards for start-clock $= 30sec$ and different play-clocks.

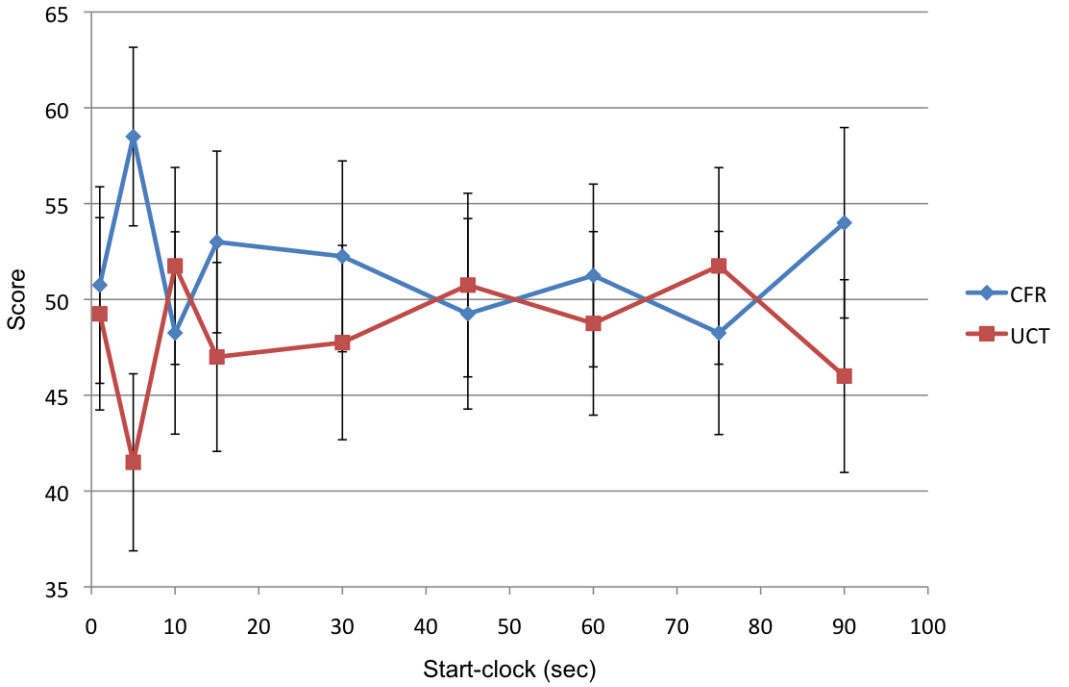


Figure 5.8: Comparing CFR and UCT in Goofspiel with 5 cards for different start-clocks and play-clock $= 10sec$.

start-clock (sec)	play-clock (sec)	number of games	CFR	UCT
30	variable	1400	52.55 ± 1.88	47.40 ± 1.88
variable	10	1800	51.10 ± 1.66	48.86 ± 1.66

Table 5.3: CFR vs. UCT comparison for different settings of start and play clocks in Goofspiel with 5 cards.

players	start-clock (min)	play-clock (sec)	player 1	player 2	player 3
CFR - UCT1 - UCT2	3.5	10	81.5	50.1	51.7
CFR - UCT1 - UCT2	10	10	79.9	46.6	50.1
UCT - CFR1 - CFR2	3.5	10	72.1	68.3	72.7
UCT - CFR1 - CFR2	10	10	50	79.8	70.6

Table 5.4: CFR vs. UCT comparison in three-player smallest.

Although the difference between the performance of CFR and UCT is not statistically significant for most of data points in either case to suggest any patterns, but the lines representing the performance of each algorithm do not cross in Figure 5.7 while they do several number of times in Figure 5.8. This may suggest that CFR is more robust to variable play-clocks. Table 5.3 gives the average score over all the games in either case with 95 percent confidence intervals. All differences are more than 95 percent statistically significant using t-test. As it can be seen, CFR is slightly stronger given a 30 second start-clock in Goofspiel with 5 cards for variable play-clock (the 95 percent confidence intervals for CFR and UCT are $(50.67, 54.43)$ and $(45.52, 49.28)$ respectively that do no intersect).

5.3.1 2009 GGP Competition

In addition to considering two-player constant-sum games (Table 5.2), we also considered a three-player version of the game called *smallest* that was used in the GGP competition in 2009. The game is fairly simple. At each step of the game, every player must announce a number from 1 to 10. The player who announces the smallest unique number will get 10 points. For example if player 1, 2, and 3 announce 1, 2, and 3 respectively, player 1 will get 10 points. However, if player 3 decides to announce 1 while player 1 and 2 still announce 1 and 2, then player 2 will get 10 points. If all the players announce the same number, no player will get any points for that round since no player announced a unique number. The game ends when the first player gets 100 points or 25 steps are passed.

The results for two different seatings in this game are given in Table 5.4. There are one CFR player and two UCT players in the first seating, while there are two CFR players and one UCT player in the second one. We initialized the probabilities in CFR players using a uniform distribution. Two different time settings were used for each seating. The shorter, *i.e.* 3.5 minutes, start-clock is set to be just enough for CFR to build the entire game tree.

It can be seen that when short start-clock is used and there is only one CFR player in the game, CFR wins (with 30 points difference in the scores). However, when there are two CFR players

	b_1	b_2
a_1	$23, 77$	$77, 23$
a_2	$73, 27$	$27, 73$

Table 5.5: The payoff matrix of a simple game.

involved, the match is basically a tie for all the players. This suggests that while the uniform initialization is good in this game (former seating), when CFR does not have enough time to do sufficient number of iterations it can do poorly (latter seating). These results are analogous to the results in the GGP competition in 2009 where random player beat every other player.

However, using a longer start-clock changes the results. In fact, a 10 minute start-clock can be considered as an example of a case where UCT player does not compete with CFR player given a long start-clock. In both seating when a 10 minute start-clock is used, the CFR players outperform the UCT players by a large margin (almost 30 points in former and 20 points in the latter seating). The reason for the strength of the CFR players in this game is that the UCT players play deterministically. Therefore, when there is another player who plays identical to a UCT player in this game, can make him suffer. This is in contrast to CFR that randomizes between its actions to gain more points when it is given *enough* time to run *enough* number of iterations. Unfortunately, we have not integrated CFR in our general game player that competed in the 2009 GGP competition. If we had done the integration, we might have won the competition.

5.4 Exploiting the Opponent

In any game that involves other agents than the player himself, knowledge about the models of the other players can be very beneficial. If models of the other players are known, then the player can adapt his strategy to gain an advantageous outcome based on his knowledge. For example, a model could be exploited to take advantage of defects in the opponent's strategy. However, just by following a Nash equilibrium strategy we will not be able to exploit an opponent's strategy.

As an example, consider the payoff matrix of a simple game shown in Table 5.5. There is only one Nash equilibrium for that game with the expected value of 50 for both players. The mixed strategy action probabilities are as follows.

$$P_1(a_1) = 0.46 \quad , \quad P_1(a_2) = 0.54$$

$$P_2(b_1) = 0.5 \quad , \quad P_2(b_2) = 0.5$$

Suppose the first player tends to select a_2 all the time. If we just follow the mixed strategy probabilities for action selection we will only get $\frac{1}{2} \times 27 + \frac{1}{2} \times 73 = 50$ points versus the potential 73 points that we could have got if we had used our knowledge about our opponent properly.

5.5 Exploiting UCT

We discussed in Section 3.6 that UCT will not necessarily converge to a Nash equilibrium. But if we just adhere to the Nash equilibrium while playing against UCT, we are only guaranteed to get the Nash equilibrium expected value. However, if we could model the probability distribution that UCT converges to, we could exploit UCT and gain more than what we could gain just by following the Nash equilibrium.

If we have a model of the probability distribution for action selection by UCT (or any other player), then exploiting UCT by CFR is straightforward. We set the probabilities for the player that UCT is going to play his role in the game equal to the probabilities that we assume UCT will use to actually play the game. Then we use CFR to compute the probabilities for the player that we will play his role in the game while keeping the probabilities for the opponent (in this case the UCT player) fixed. Finally we use the new probabilities to play the game. This approach will result in a best response to the probability setting assumed for UCT. However, since it is an open question as to what solution UCT converges to in simultaneous move games in general and the distribution of probabilities is not known in advance, we cannot compute the best response to UCT for every game. In addition, the opponent may not be even using UCT. Therefore, using the best response approach can be brittle and can suffer greatly if the assumed model is wrong.

To address the latter problem, it is desirable to exploit a known opponent but still be close to a Nash equilibrium so as to not be exploitable to a large extent. Two approaches can be taken to exploit an opponent and still not suffer greatly if the model is wrong. One of them is to compute both the best response and a mixed strategy Nash equilibrium and alternate between them. We can assume different probabilities for using each of the probability distributions to achieve different levels of trade-off between exploitation and exploitability. Another approach is to assume that with probability p our opponent adheres to what we assumed, and with probability $1 - p$ the player tries to minimize his regret and play a Nash equilibrium. Afterward, we can use this new model of the opponent to compute a mixed strategy Nash equilibrium to play the game (this new equilibrium is called a restricted Nash equilibrium). Different variations of p can lead to different levels of exploitation and exploitability. In poker the latter approach has been shown to be superior to the former [18].

The results for computing the restricted Nash equilibrium for biased rock-paper-scissors (discussed in Section 3.6.1) are given in Table 5.6. The first column shows how much we trust our model (different values of p). The second column, named exploitation, shows the score for different degrees of UCT exploitation by CFR. The last column, named exploitability, shows the score if the CFR player which is exploiting the UCT player happens to play against the best response to itself. The results are averaged over 100 games. A 30 second start-clock and 10 second play-clock was used. As our model of the opponent, we assumed that UCT will select rock all the times. The difference between the scores of CFR and UCT are more than 99 percent statistically significant when

model confidence (p)	exploitation (CFR vs. UCT)	exploitability (CFR vs. best response)
1	$75 - 25$	$45 - 55$
0.75	$75 - 25$	$45 - 55$
0.5	$74.5 - 25.5$	$49.5 - 50.5$
0.25	$73.25 - 26.75$	$50.15 - 49.85$
0	$50.25 - 49.75$	$52.35 - 47.65$

Table 5.6: Exploitation vs. exploitability in biased rock-paper-scissors.

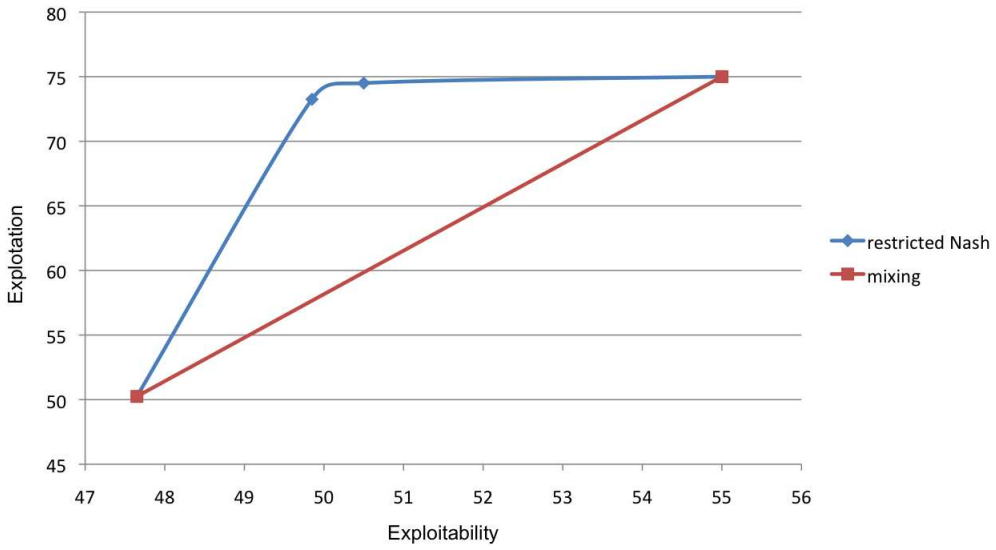


Figure 5.9: Exploitation vs. exploitability in biased rock-paper-scissors.

we are using a model ($p > 0$) using the t-test. It can be seen that if we have access to the model of the opponent ($p > 0$), we can benefit greatly. However, if we rely too much on the model and we happen to play against an opponent that knows about our assumed model, we can suffer badly[2] ($p = 0.75$ and $p = 1$).

If we consider the best response ($p = 1$) and the mixed strategy Nash equilibrium, we can achieve any exploitation and exploitability trade-off by different mixing in between. The straight line between $(47.65, 50.25)$ and $(55, 75)$ in Figure 5.9 corresponds to this mixing.

Using these two approaches, we will be in a safe margin if our model of the opponent happens to be wrong. As can be seen in Figure 5.9 the curve for the restricted Nash equilibrium approach is above the line of the mixing approach for biased rock-paper-scissors. This means that in the CFR algorithm, if we give up a small amount for being exploitable we can exploit a UCT player a lot.

The results for computing the restricted Nash equilibrium for Goofspiel with 5 cards are given in Table 5.7. This results are also averaged over 100 games. A one minute start-clock and a 20 second play-clock was used. As our model of the opponent, we assumed that UCT will select the action with the highest probability at each step. It must be mentioned that our assumption about the model

[2] 10 points is the most that can be suffered according to the payoff matrix (Table 3.3) and we suffered that much by relying too much on the model.

model confidence (p)	exploitation (CFR vs. UCT)	exploitability (CFR vs. best response)
1	$77 - 23$	$0 - 100$
0.75	$60.5 - 39.5$	$15.5 - 84.5$
0.5	$51 - 49$	$37.5 - 62.5$
0.25	$48 - 52$	$53 - 47$
0	$50 - 50$	$50 - 50$

Table 5.7: Exploitation vs. exploitability in 5 cards Goofspiel.

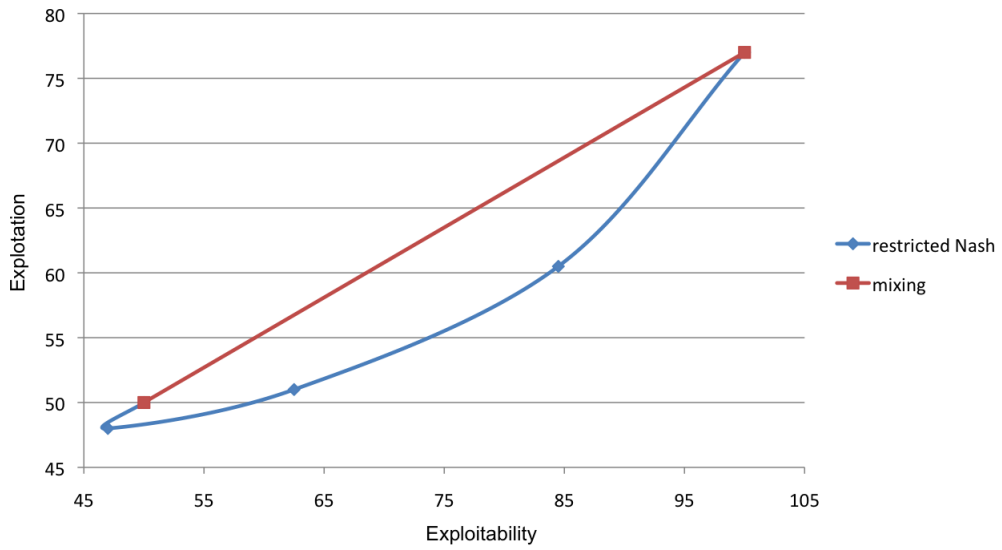


Figure 5.10: Exploitation vs. exploitability in Goofspiel with 5 cards.

of the UCT player can be wrong. It can be seen that the best response to UCT player ($p = 1$) can exploit UCT the most (77 to 23), but is very exploitable itself (losing 100 to 0). However, we cannot exploit UCT very much by not relying on the model ($p = 0.25$), but we will not be exploitable either.

If we consider mixing between the best response ($p = 1$) and the mixed strategy Nash equilibrium, we can achieve any exploitation and exploitability trade-off between their extreme points. The straight line between $(50, 50)$ and $(100, 77)$ in Figure 5.10 corresponds to this mixing. Although the restricted Nash equilibrium method outperformed the mixing approach in biased rock-paper-scissors, but as it can be seen in Figure 5.10 the curve for the restricted Nash equilibrium approach is below the line of mixing approach for Goofspiel with 5 cards. Therefore it is better to use the mixing approach in this game. This can be due to the fact that our model of the UCT in this game is wrong. In addition, it must be considered although our assumption about the model of the UCT player can be wrong, but we use the right model when we are computing the best response to the CFR that is exploiting UCT, which is not available to the players in the real competition. However, the nice property of giving up a small amount of exploitability and exploiting to a large extent is not true in this game and the best that we can do using the perceived model is the linear trade-off in the

mixing approach.

Therefore an inaccurate opponent model can lead to poor performance in a restricted Nash equilibrium. In addition, the quality of the model has a direct effect on the performance, independent of the method we use to exploit that model. Thus we will be better off if we have a better model of our opponent. However, since we still do not have a model for UCT in simultaneous move games and we do not know the characteristics of the balanced situation that UCT converges to, trying to build a good model and defining the characteristics of the balanced situation is promising future work.

5.6 Conclusion

In this chapter we gave experimental result showing how different settings can effect a CFR player. We also provided experimental result showing how CFR performs against UCT. We gave an example that using UCT can be disadvantageous while using CFR would be very beneficial. In addition, we explained why following a Nash equilibrium may not be the best strategy in a situation. We also described how to exploit an opponent with a known model by using CFR while we do not endanger ourselves into being exploitable.

Chapter 6

Conclusion and Future Work

In this thesis we focused on simultaneous move games in GGP. We argued that plain UCT is not sufficient to be used in simultaneous move games. We explained a variant of the UCT algorithm to be used in this class of games. We discussed that even the enhanced variant of UCT does not converge to Nash equilibrium in general.

We demonstrated the CFR algorithm and explained how it can be used in GGP. We discussed the simplifications and challenges that face the use of CFR in GGP. We also discussed why it is advantageous to use CFR instead of UCT for solving simultaneous move games. We gave experimental results for a game used in the last general game playing competition that showed how using UCT can make an agent suffer greatly.

In addition, we explained why following a Nash equilibrium may not be the best strategy all the times. We also discussed how UCT can be exploited by CFR.

We did not develop a model for UCT neither characterized the balanced situation that it converges to. It is an open question that if it is possible to derive a model for how UCT plays in simultaneous move games or characterize the balanced situation that it converges to. Having the model or the characteristic of the balanced situation, CFR can be used easily to exploit it as explained in Chapter 5.

We did not propose a smart way of tree expansion for CFR. A beneficial future work is to define how time management should be done in a CFR player and what portion of time should be devoted to tree expansion, simulations, and iterations.

Another interesting future work is to solve simultaneous move games with Linear Programming (LP). Since all the games in GGP are perfect information games, LP can be used to solve simultaneous move games using a bottom-up fashion in the game tree. Solving the problem using LP starts at the leaf nodes and the result of the subgame just solved will be used to solve the subgame just before that in the game tree after all the subgames of the parent subgame are solved. This approach has the benefit of being accurate versus the incremental approach of CFR, but the player must have enough time to complete one sweep of the whole game tree.

Simultaneous move games are basically more complex than sequential games, because of the

higher branching factor resulted from consideration of all the different combination of moves of different players. Therefore solving simultaneous move games and playing acceptable requires longer start-clock and play-clock. In addition, in simultaneous move games we deal with probabilities and playing a game only once cannot tell anything about the strength of a player. Thus, to get a more statistically significant result in the competition, we hope to see that a game is played more than just once.

Finally if we do any opponent modeling, it will be very beneficial to know the player that we are playing against so that we can use our models. Therefore, providing the information about the players that are playing a game during the start-clock can be advantageous and omit the need for player detection in addition to opponent modeling.

Bibliography

[1] Nima Asgharbeygi, David Stracuzzi, and Pat Langley. Relational temporal difference learning. In *ICML '06: International Conference on Machine Learning*, pages 49–56, 2006.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[3] Bikramjit Banerjee, Gregory Kuhlmann, and Peter Stone. Value function transfer for general game playing. In *ICML workshop on Structural Knowledge Transfer for Machine Learning*, June 2006.

[4] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *International Joint Conference on Artificial Intelligence*, January 2007.

[5] Yngvi Björnsson and Hilmar Finnsson. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1:4–15, March 2009.

[6] Murray Campbell, A. Joseph Hoane Jr., and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.

[7] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *AAAI, Vol. 1*, pages 120–125, 1996.

[8] James Clune. Heuristic evaluation functions for general game playing. In *AAAI*, pages 1134–1139, 2007.

[9] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pages 72–83, 2006.

[10] Evan Cox, Eric Schkufza, Ryan Madsen, and Michael Genesereth. Factoring general game using propositional automata. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 13–20, 2009.

[11] H. H. L. M. Donkers, H. Jaap van den Herik, and Jos W. H. M. Uiterwijk. Selecting evaluation functions in opponent-model search. *Theoretical Computer Science*, 349(2):245–267, 2005.

[12] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, pages 259–264, 2008.

[13] Hilmar Finnsson and Yngvi Björnsson. Simulation control in general game playing agents. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 21–26, 2009.

[14] Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *The NIPS On-line trading of Exploration and Exploitation Workshop*, December 2006.

[15] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France, November 2006.

[16] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI Magazine*, 26(2):62–72, 2005.

[17] Marting Gunther, Stephan Schiffel, and Michael Thiescher. Factoring general games. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 27–24, 2009.

[18] Michael Johanson, Martin Zinkevich, and Michael Bowling. Computing robust counter-strategies. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. MIT Press, 2007.

[19] David M. Kaiser. Automatic feature extraction for autonomous general game playing agents. In *Autonomous Agents and Multiagent Systems*, pages 1–7, 2007.

[20] Mesut Kirci, Jonathan Schaeffer, and Nathan Sturtevant. Feature learning using state differences. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 35–42, 2009.

[21] Peter Kissmann and Stefan Edelkamp. Instantiating general games. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 43–50, 2009.

[22] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[23] Gregory Kuhlmann and Peter Stone. Automatic heuristic construction in a complete general game player. In *AAAI*, pages 1457–1462, 2006.

[24] Gregory Kuhlmann and Peter Stone. Graph-based domain mapping for transfer learning in general games. In *European Conference on Machine Learning*, pages 188–200, September 2007.

[25] Robert Levinson. General game-playing and reinforcement learning. *Computational Intelligence*, 12:155–176, 1996.

[26] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. *General Game Playing: Game Description Language Specification*. Stanford Logic Group, Computer Science Department, Stanford University, http://games.stanford.edu/language/spec/gdl_spec_2008_03.pdf, March 2008.

[27] Barney Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, University of Cambridge, 1993.

[28] Joseph Reisinger, Erkin Bahceci, Igor Karpov, and Risto Mukkulainen. Coevolving strategies for general game playing. In *Computational Intelligence and Games, 2007*, pages 320–327, 2007.

[29] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice. *Pattern Analysis and Machine Intelligence*, 11(11):1203–1212, 1989.

[30] Jonathan Schaeffer, Yngvi Björnsson, Neil Burch, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Solving checkers. In *IJCAI*, pages 292–297, 2005.

[31] Stephan Schiffel. Symmetry detection in general game playing. *The IJCAI Workshop on General Game Playing (GIGA'09)*, pages 67–74, 2009.

[32] Stephan Schiffel and Michael Thielscher. Automatic construction of a heuristic search function for general game playing. In *IJCAI Workshop on Nonmontonic Reasoning, Action and Change (NRAC07)*, Hyderabad, India, 2007.

[33] Stephan Schiffel and Michael Thielscher. Automated theorem proving for general game playing. In *International Joint Conference on Artificial Intelligence*, pages 911–916, 2009.

[34] Shiven Sharma, Ziad Kobti, and Scott D. Goodwin. Knowledge generation for improving simulations in uct for general game playing. In *Australasian Conference on Artificial Intelligence*, pages 49–55, 2008.

[35] Michael Sipser. *Introduction to the Theory of Computation, Second Edition*. Course Technology, February 2005.

[36] Nathan R. Sturtevant and Richard E. Korf. On pruning techniques for multi-player games. In *AAAI*, pages 201–207, 2000.

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[38] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *NIPS*, 2007.

Appendix A

Tic-tac-toe GDL

```
; players
(role xplayer)
(role oplayer)

; initial state
(init (cell 1 1 b))
(init (cell 1 2 b))
(init (cell 1 3 b))
(init (cell 2 1 b))
(init (cell 2 2 b))
(init (cell 2 3 b))
(init (cell 3 1 b))
(init (cell 3 2 b))
(init (cell 3 3 b))
(init (control xplayer))

; player moves
(<= (next (cell ?m ?n x))
    (does xplayer (mark ?m ?n))
    (true (cell ?m ?n b)))
(<= (next (cell ?m ?n o))
    (does oplayer (mark ?m ?n))
    (true (cell ?m ?n b)))


; game axioms
(<= (next (cell ?m ?n ?w))
    (true (cell ?m ?n ?w))
    (distinct ?w b))
(<= (next (cell ?m ?n b))
    (does ?w (mark ?j ?k))
    (true (cell ?m ?n b))
    (or (distinct ?m ?j)
    (distinct ?n ?k)))
(<= (next (control xplayer))
    (true (control oplayer)))
(<= (next (control oplayer))
    (true (control xplayer)))
    (<= (row ?m ?x)
    (true (cell ?m 1 ?x))
    (true (cell ?m 2 ?x))
    (true (cell ?m 3 ?x)))
```

```
; game concepts
(<= (column ?n ?x)
    (true (cell 1 ?n ?x))
    (true (cell 2 ?n ?x))
    (true (cell 3 ?n ?x)))
(<= (diagonal ?x)
    (true (cell 1 1 ?x))
    (true (cell 2 2 ?x))
    (true (cell 3 3 ?x)))
(<= (diagonal ?x)
    (true (cell 1 3 ?x))
    (true (cell 2 2 ?x))
    (true (cell 3 1 ?x)))
(<= (line ?x)
    (row ?m ?x))
(<= (line ?x)
    (column ?m ?x))
(<= (line ?x)
    (diagonal ?x))
(<= open (true (cell ?m ?n b)))

; player moves
(<= (legal ?w (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?w)))
(<= (legal xplayer noop)
    (true (control oplayer)))
(<= (legal oplayer noop)
    (true (control xplayer)))

; goals
(<= (goal xplayer 100)
    (line x))
(<= (goal xplayer 50)
    (not (line x))
    (not (line o))
    (not open))
(<= (goal xplayer 0)
    (line o))
(<= (goal oplayer 100)
    (line o))
(<= (goal oplayer 50)
    (not (line x))
    (not (line o))
    (not open))
(<= (goal oplayer 0)
    (line x))

; terminal conditions
(<= terminal (line x))
(<= terminal (line o))
(<= terminal (not open))
```

Appendix B

Repeated Rock-paper-scissors GDL

```
(role first)
(role second)

(init (counter 0))
(<= (init (score ?player 0))
    (role ?player))

(<= (next (counter ?x2))
    (true (counter ?x1))
    (successor ?x1 ?x2))

(<= (next (score ?player ?x2))
    (role ?player)
    (true (score ?player ?x1))
    (successor ?x1 ?x2)
    (win ?player ?opponent)
    (role ?opponent)
    (distinct ?player ?opponent))
(<= (next (score ?player ?x1))
    (role ?player)
    (true (score ?player ?x1))
    (not (win ?player ?opponent))
    (role ?opponent)
    (distinct ?player ?opponent))

(<= (legal ?player rock)
    (role ?player))
(<= (legal ?player paper)
    (role ?player))
(<= (legal ?player scissors)
    (role ?player))

(<= (goal ?player 100)
    (role ?player)
    (true (score ?player ?x1))
    (true (score ?opponent ?x2))
    (role ?opponent)
    (distinct ?player ?opponent)
    (greater ?x1 ?x2))
(<= (goal ?player 50)
    (role ?player)
    (true (score ?player ?x1))
    (true (score ?opponent ?x1))
```

```
    (role ?opponent)
    (distinct ?player ?opponent))
(<= (goal ?player 0)
    (role ?player)
    (true (score ?player ?x1))
    (true (score ?opponent ?x2))
    (role ?opponent)
    (distinct ?player ?opponent)
    (greater ?x2 ?x1))

(<= terminal
    (counter 9))

(successor 0 1)
(successor 1 2)
(successor 2 3)
(successor 3 4)
(successor 4 5)
(successor 5 6)
(successor 6 7)
(successor 7 8)
(successor 8 9)

(<= (greater ?x1 ?x2)
    (successor ?x2 ?x1))
(<= (greater ?x1 ?x2)
    (successor ?x3 ?x1)
    (greater ?x3 ?x2))

(<= (win ?player ?opponent)
    (does ?player rock)
    (does ?opponent scissors))
(<= (win ?player ?opponent)
    (does ?player paper)
    (does ?opponent rock))
(<= (win ?player ?opponent)
    (does ?player scissors)
    (does ?opponent paper))
```

Appendix C

Goofspiel with 5 Cards GDL

```
(role first)
(role second)

(init (current g1))
(init (lastwinner NULL))
(init (position g1 c1))
(init (position g2 c2))
(init (position g3 c3))
(init (position g4 c4))
(init (position g5 c5))

(<= (init (points ?player 0))
    (role ?player))

(<= (init (hand ?player c1))
    (role ?player))
(<= (init (hand ?player c2))
    (role ?player))
(<= (init (hand ?player c3))
    (role ?player))
(<= (init (hand ?player c4))
    (role ?player))
(<= (init (hand ?player c5))
    (role ?player))

(<= (next (current ?g2))
    (true (current ?g1))
    (succ ?g1 ?g2))

(<= (next (position ?g ?c))
    (true (position ?g ?c)))

(<= (legal ?player (drop ?c))
    (role ?player)
    (true (hand ?player ?c)))

(<= (next (hand ?player ?c))
(role ?player)
    (true (hand ?player ?c))
    (does ?player (drop ?cc))
    (distinct ?c ?cc))

(<= (next (lastwinner ?player))
```

```
(role ?player)
(win ?player ?opponent)
(role ?opponent)
(distinct ?player ?opponent))

(<= (next (lastwinner NULL))
(role ?player)
(not (win ?player ?opponent))
(role ?opponent)
(distinct ?player ?opponent)
(not (win ?opponent ?player)))

(<= (next (points ?player ?new))
    (role ?player)
    (true (points ?player ?old))
    (win ?player ?opponent)
    (role ?opponent)
    (distinct ?player ?opponent)
    (true (current ?g))
    (true (position ?g ?c))
    (value ?c ?delta)
    (diff ?new ?old ?delta))

(<= (next (points ?player ?old))
    (role ?player)
    (true (points ?player ?old))
    (not (win ?player ?opponent))
    (role ?opponent)
    (distinct ?player ?opponent))

(<= (diff ?v2 ?v1 ?delta)
    (inc ?v1 ?v2 ?delta))

(<= (diff ?v3 ?v1 ?delta)
(successor ?v2 ?v3)
(successor ?dd ?delta)
(inc ?v2 ?v3 1)
(diff ?v2 ?v1 ?dd))

(<= terminal
    (true (current g6)))

(<= (goal ?player 100)
    (role ?player)
    (true (current g6))
    (true (points ?player ?p1))
    (true (points ?opponent ?p2))
    (distinct ?player ?opponent)
    (greater ?p1 ?p2))

(<= (goal ?player 0)
    (role ?player)
    (true (current g6))
    (true (points ?player ?p1))
    (true (points ?opponent ?p2))
    (distinct ?player ?opponent)
    (greater ?p2 ?p1))

(<= (goal ?player 50)
```

```
    (role ?player)
    (true (current g6))
    (true (points ?player ?p))
    (true (points ?opponent ?p))
    (distinct ?player ?opponent))

(<= (greater ?c1 ?c2)
    (successor ?c1 ?c2))

(<= (greater ?c1 ?c3)
    (successor ?c1 ?c2)
    (greater ?c2 ?c3))

(value c1 1)
(value c2 2)
(value c3 3)
(value c4 4)
(value c5 5)

(succ g1 g2)
(succ g2 g3)
(succ g3 g4)
(succ g4 g5)
(succ g5 g6)

(successor 0 1)
(successor 1 2)
(successor 2 3)
(successor 3 4)
(successor 4 5)
(successor 5 6)
(successor 6 7)
(successor 7 8)
(successor 8 9)
(successor 9 10)
(successor 10 11)
(successor 11 12)
(successor 12 13)
(successor 13 14)
(successor 14 15)

(inc 0 1 1)
(inc 1 2 1)
(inc 2 3 1)
(inc 3 4 1)
(inc 4 5 1)
(inc 5 6 1)
(inc 6 7 1)
(inc 7 8 1)
(inc 8 9 1)
(inc 9 10 1)
(inc 10 11 1)
(inc 11 12 1)
(inc 12 13 1)
(inc 13 14 1)
(inc 14 15 1)

(<= (win ?player ?opponent)
    (does ?player (drop c5))
```

```
    (does ?opponent (drop c1)))
(<= (win ?player ?opponent)
    (does ?player (drop c5))
    (does ?opponent (drop c2)))
(<= (win ?player ?opponent)
    (does ?player (drop c5))
    (does ?opponent (drop c3)))
(<= (win ?player ?opponent)
    (does ?player (drop c5))
    (does ?opponent (drop c4)))

(<= (win ?player ?opponent)
    (does ?player (drop c4))
    (does ?opponent (drop c1)))
(<= (win ?player ?opponent)
    (does ?player (drop c4))
    (does ?opponent (drop c2)))
(<= (win ?player ?opponent)
    (does ?player (drop c4))
    (does ?opponent (drop c3)))

(<= (win ?player ?opponent)
    (does ?player (drop c3))
    (does ?opponent (drop c1)))
(<= (win ?player ?opponent)
    (does ?player (drop c3))
    (does ?opponent (drop c2)))

(<= (win ?player ?opponent)
    (does ?player (drop c2))
    (does ?opponent (drop c1)))
```

Appendix D

Simultaneous Breakthrough GDL

```
(role white)
(role black)

(init (cellholds 1 1 white))
(init (cellholds 2 1 white))
(init (cellholds 3 1 white))
(init (cellholds 1 5 black))
(init (cellholds 2 5 black))
(init (cellholds 3 5 black))

(init (dominant white))

(<= (legal white (move ?x ?y1 ?x ?y2))
    (true (cellholds ?x ?y1 white))
    (plusplus ?y1 ?y2)
    (cellempty ?x ?y2))
(<= (legal white (move ?x1 ?y1 ?x2 ?y2))
    (true (cellholds ?x1 ?y1 white))
    (cell ?x2 ?y2)
    (plusplus ?y1 ?y2)
    (plusplus ?x1 ?x2)
    (not (true (cellholds ?x2 ?y2 white))))
(<= (legal white (move ?x1 ?y1 ?x2 ?y2))
    (true (cellholds ?x1 ?y1 white))
    (cell ?x2 ?y2)
    (plusplus ?y1 ?y2)
    (plusplus ?x2 ?x1)
    (not (true (cellholds ?x2 ?y2 white))))

(<= (legal black (move ?x ?y1 ?x ?y2))
    (true (cellholds ?x ?y1 black))
    (plusplus ?y2 ?y1)
    (cellempty ?x ?y2))
(<= (legal black (move ?x1 ?y1 ?x2 ?y2))
    (true (cellholds ?x1 ?y1 black))
    (cell ?x2 ?y2)
    (plusplus ?y2 ?y1)
    (plusplus ?x1 ?x2)
    (not (true (cellholds ?x2 ?y2 black))))
(<= (legal black (move ?x1 ?y1 ?x2 ?y2))
    (true (cellholds ?x1 ?y1 black))
    (cell ?x2 ?y2)
    (plusplus ?y2 ?y1)
```

```
    (plusplus ?x2 ?x1)
    (not (true (cellholds ?x2 ?y2 black))))

(<= (next (dominant white))
    (true (dominant black))
    (does white (move ?x1 ?y1 ?x2 ?y2))
    (does black (move ?x3 ?y3 ?x2 ?y2)))
(<= (next (dominant black))
    (true (dominant white))
    (does white (move ?x1 ?y1 ?x2 ?y2))
    (does black (move ?x3 ?y3 ?x2 ?y2)))
(<= (next (dominant white))
    (true (dominant white))
    (does white (move ?x1 ?y1 ?x2 ?y2))
    (does black (move ?x4 ?y4 ?x3 ?y3))
    (distinctcell ?x2 ?y2 ?x3 ?y3))
(<= (next (dominant black))
    (true (dominant black))
    (does white (move ?x1 ?y1 ?x2 ?y2))
    (does black (move ?x4 ?y4 ?x3 ?y3))
    (distinctcell ?x2 ?y2 ?x3 ?y3))

(<= (next (cellholds ?x2 ?y2 ?player))
    (role ?player)
    (does ?player (move ?x1 ?y1 ?x2 ?y2))
    (role ?opponent)
    (does ?opponent (move ?x3 ?y3 ?x4 ?y4))
    (distinct ?player ?opponent)
    (distinctcell ?x2 ?y2 ?x4 ?y4))
(<= (next (cellholds ?x2 ?y2 ?player))
    (role ?player)
    (does ?player (move ?x1 ?y1 ?x2 ?y2))
    (role ?opponent)
    (does ?opponent (move ?x3 ?y3 ?x2 ?y2))
    (distinct ?player ?opponent)
    (dominant ?player))

(<= (next (cellholds ?x5 ?y5 ?state))
    (true (cellholds ?x5 ?y5 ?state))
    (does white (move ?x1 ?y1 ?x2 ?y2))
    (does black (move ?x4 ?y4 ?x3 ?y3))
    (distinctcell ?x1 ?y1 ?x5 ?y5)
    (distinctcell ?x2 ?y2 ?x5 ?y5)
    (distinctcell ?x3 ?y3 ?x5 ?y5)
    (distinctcell ?x4 ?y4 ?x5 ?y5))

(<= terminal
    whitewin)
(<= terminal
    blackwin)
(<= terminal
    (role ?player)
    (nocell ?player))

(<= (goal white 100)
     whitewin
    (not blackwin))
(<= (goal white 100)
    (nocell black)
```

```
    (true (cellholds ?x ?y white)))
(<= (goal white 50)
     whitewin
     blackwin)
(<= (goal white 0)
    (not whitewin))
(<= (goal white 0)
    (nocell white))
(<= (goal black 100)
     blackwin
    (not whitewin))
(<= (goal black 100)
    (nocell white)
    (true (cellholds ?x ?y black)))
(<= (goal black 50)
     whitewin
     blackwin)
(<= (goal black 0)
    (not blackwin))
(<= (goal black 0)
    (nocell black))

(<= (cell ?x ?y)
    (x ?x)
    (y ?y))
(<= (cellempty ?x ?y)
    (cell ?x ?y)
    (not (true (cellholds ?x ?y white)))
    (not (true (cellholds ?x ?y black))))
(<= (distinctcell ?x1 ?y1 ?x2 ?y2)
    (cell ?x1 ?y1)
    (cell ?x2 ?y2)
    (distinct ?x1 ?x2))
(<= (distinctcell ?x1 ?y1 ?x2 ?y2)
    (cell ?x1 ?y1)
    (cell ?x2 ?y2)
    (distinct ?y1 ?y2))

(<= (nocell ?player)
    (role ?player)
    (not (true (cellholds 1 1 ?player)))
    (not (true (cellholds 2 1 ?player)))
    (not (true (cellholds 3 1 ?player)))
    (not (true (cellholds 1 2 ?player)))
    (not (true (cellholds 2 2 ?player)))
    (not (true (cellholds 3 2 ?player)))
    (not (true (cellholds 1 3 ?player)))
    (not (true (cellholds 2 3 ?player)))
    (not (true (cellholds 3 3 ?player)))
    (not (true (cellholds 1 4 ?player)))
    (not (true (cellholds 2 4 ?player)))
    (not (true (cellholds 3 4 ?player)))
    (not (true (cellholds 1 5 ?player)))
    (not (true (cellholds 2 5 ?player)))
    (not (true (cellholds 3 5 ?player))))

(<= whitewin
    (x ?x)
    (true (cellholds ?x 5 white)))
```

```
(<= blackwin
    (x ?x)
    (true (cellholds ?x 1 black)))

(plusplus 1 2)
(plusplus 2 3)
(plusplus 3 4)
(plusplus 4 5)
(x 1)
(x 2)
(x 3)
(y 1)
(y 2)
(y 3)
(y 4)
(y 5)
```