FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Framework for Monte Carlo Tree Search-related strategies in Competitive Card Based Games

Pedro Ricardo Oliveira Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira (PhD)

Second Supervisor: Ivo Timóteo (MSc)

June 27, 2016

Framework for Monte Carlo Tree Search-related strategies in Competitive Card Based Games

Pedro Ricardo Oliveira Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. dr. Henrique Lopes Cardoso

External Examiner: Prof. dr. Pedro Ribeiro (Universidade do Porto, FCUP)

Supervisor: Prof. dr. Hugo Sereno Ferreira

June 27, 2016

Abstract

In recent years, Monte Carlo Tree Search (MCTS) has been successfully applied as a new artificial intelligence strategy in game playing, with excellent results yielded in the popular board game "Go", real time strategy games and card games. The MCTS algorithm was developed as an alternative over established adversarial search algorithms, e.g., Minimax (MM) and knowledge-based approaches.

MCTS can achieve good results with nothing more than information about the game rules and can achieve breakthroughs in domains of high complexity, whereas in traditional AI approaches, developers might struggle to find heuristics through expertise in each specific game.

Every algorithm has its caveats and MCTS is no exception, as stated by Browne et al: "Although basic implementations of MCTS provide effective play for some domains, results can be weak if the basic algorithm is not enhanced. (...) There is currently no better way than a manual, empirical study of the effect of enhancements to obtain acceptable performance in a particular domain." [BPW+12]

Thus, the first objective of this dissertation is to research various state of the art MCTS enhancements in a context of card games and then proceed to apply, experiment and fine tune them in order to achieve a highly competitive implementation, validated and tested against other algorithms such as MM.

By analysing trick-taking card games such as Sueca and Bisca, where players take turns placing cards face up in the table, there are similarities which allow the development of a MCTS based implementation that features enhancements effective in multiple game variations, since they are non deterministic imperfect information problems. Good results have been achieved in this domain with the algorithm, in games such as Spades [WCPR13a] and Skat [FB13].

The end result aims toward a framework that offers a competitive AI implementation for the card games Sueca, Bisca and Hearts (achieved with analysis and validation against other approaches), allowing developers to integrate their own card games and benefit from a working AI, and also serving as testing ground to rank different agent implementations.

Resumo

Nos últimos anos, o algoritmo de *Monte Carlo Tree Search* (MCTS) tem sido aplicado com sucesso como uma nova abordagem de inteligência artificial (IA) em jogos, obtendo excelentes resultados no jogo de tabuleiro "Go", jogos de estratégia em tempo real e, também, jogos de cartas. O MCTS foi desenvolvido como uma alternativa a algoritmos clássicos de pesquisa adversarial (como, por exemplo, *Minimax* (MM) e abordagens baseadas em regras de conhecimento de jogo).

O MCTS consegue alcançar bons resultados apenas com a declaração de regras de jogo, permitindo, assim, o avanço do estado da arte em inteligência artificial para domínios de alta complexidade, enquanto que as abordagens tradicionais de IA não prescidem do uso de heurísticas com base em conhecimento do domínio. Em alguns casos, as heurísticas são complexas de formular, inviabilizando o uso destas abordagens.

No entanto, a aplicação de MCTS também apresenta as suas desvantagens, nomeadamente: os resultados são geralmente considerados fracos se a versão básica do algoritmo não for melhorada. Para obter um desempenho aceitável num domínio, não existe, atualmente, melhor alternativa do que o estudo manual e empírico dos efeitos de melhorias no algoritmo [BPW+12].

Consequentemente, o primeiro objetivo desta dissertação é elaborar um estudo do estado da arte de melhorias para o MCTS, aplicadas ao domínio de jogos de cartas. A experimentação com melhorias permite obter uma implementação competitiva que será avalidada e testada contra outros algoritmos como, por exemplo, o MM.

Jogos de cartas em vaza, onde cada jogador coloca cartas sequencialmente no centro da mesa, como a Sueca e a Bisca, serão o foco deste estudo. Existem semelhanças nestes jogos que permitem a partilha de melhorias numa abordagem MCTS. A aplicação deste algoritmo já alcançou bons resultados no domínio de jogos de cartas, nomeadamente em Espadas [WCPR13a] e Skat [FB13].

O objetivo final é desenvolver uma *framework* que oferece uma implementação de IA para os jogos de cartas Sueca, Bisca e Copas, com análise e validação contra diferentes abordagens. A *framework* irá permitir a programadores ou investigadores a integração dos seus próprios jogos de cartas, servindo como um ponto de partida para testar e avaliar a eficiência de diferentes implementações de agentes de IA.

Acknowledgements

Firstly I would like to thank both my supervisors, Prof. Hugo Sereno and Ivo Timóteo, for offering me this challenge, which sparkled my interest in the field of artificial intelligence, and also for their guidance and support throughout this dissertation.

I want to thank my girlfriend, Isabel, who accompanied me through all my years of study and work, including both good and bad times. Her love and patience has always been my tower of strength, and with her support, I am always ready to embrace tougher challenges every day.

I also want to thank my family, especially my parents, for their love and support, as well as my brother, Luís, who never hesitated to help whenever I needed the most, always ready and willing to share his knowledge and lend a hand.

I would also like to thank my company, Blip, for providing me with excellent working conditions while studying, and for always letting me prioritize studies when necessary. I specially thank my co-workers from the Avengers Team, namely, Bárbara Salgado, Gonçalo Matos, Hugo Bacelar, João Cunha, Nuno Estrada, Pedro Santos, Sandrine Mendes, Tiago Alves and Tiago Morais. The team accompanied me through every step of this journey, always sharing my triumphs and worries. Working with them was an experience that I will personally treasure and be thankful for.

Last, but not least, I would like to thank my friends, André and Débora, for their care and attention. They were always present through many hard times, and we are thankful for having them as our second family.

Pedro Fernandes

"Imagination is more important than knowledge.
For knowledge is limited, whereas imagination embraces the entire world,
stimulating progress, giving birth to evolution.
It is, strictly speaking, a real factor in scientific research."

Albert Einstein

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Problem Definition . . . 2
1.3 Motivation and Goals . . . 3
1.4 Report Structure . . . 3

2 Related Work **5**
2.1 Decision Theory . . . 5
2.2 Game Theory . . . 6
2.3 Multi Armed Bandits . . . 7
2.4 Monte Carlo Tree Search . . . 8
2.4.1 Properties of MCTS . . . 9
2.4.2 Upper Confidence Bounds for Trees (UCT) . . . 10
2.4.3 Enhancements . . . 11
2.5 Adversarial Search in Imperfect Information Games . . . 14
2.5.1 Perfect Information Monte Carlo . . . 14
2.5.2 Minimax . . . 18
2.5.3 Information Set MCTS . . . 18

3 Trick Taking Card Games **23**
3.1 Main Characteristics . . . 23
3.2 Sueca . . . 24
3.3 Bisca . . . 25
3.4 Hearts . . . 26

4 Implementation **29**
4.1 Determinization Algorithms . . . 29
4.2 Botwars Framework . . . 37
4.3 MCTS Framework for Trick Taking Card Games . . . 39

5 MCTS Algorithms applied to Trick Taking Card Games **43**
5.1 Determinized UCT . . . 43
5.2 ISMCTS . . . 46
5.3 Reward Functions . . . 49
5.4 NAST Simulation Enhancement . . . 52
5.4.1 Simulation Policies . . . 53
5.5 Minimax . . . 55

6 Experimental Results **57**
6.1 Determinized UCT versus ISMCTS 58
6.2 Number of ISMCTS Iterations 59
6.3 Random and Cheating Play 60
6.4 Reward Functions 62
6.5 NAST Enhancement 64
6.6 Hybrid ISMCTS with Minimax 67

7 Conclusions **69**
7.1 Goals 69
7.2 Future Research 72

References **75**

List of Figures

2.1 One iteration of the general MCTS approach 8
2.2 MCTS assymetric tree growth 10
2.3 A representation of node correlation for various MCTS enhancements 12
2.4 Average playing strength results for different combinations of MCTS enhancements 13
2.5 Playing strengths for Dou Di Zhu and Hearts, with EPIC and NAST 13
2.6 Examples of strategy fusion and non-locality 16
2.7 Performance gain of PIMC search over random against a Nash equilibrium . . . 17
2.8 An example of Determinized UCT search trees 19
2.9 An example of ISMCTS search restriction in the game tree 19

4.1 The botwars client UI while playing a Sueca game 39
4.2 The MCTS framework tree visualisation tool 41

6.1 Results for Determinized UCT against ISMCTS 59
6.2 Results for ISMCTS iteration number 60
6.3 Results for random and cheating play 61
6.4 Results for reward function experiment 62
6.5 Results for best UCB1 exploration constant using Scores Difference reward function 63
6.6 Results for best UCB1 exploration constant using Win or Loss reward function . 64
6.7 Results for best NAST simulation policy 65
6.8 Results for best NAST N-gram length 66
6.9 Results for hybrid use of Minimax and ISMCTS 67

LIST OF FIGURES

Abbreviations

MCTS	Monte Carlo Tree Search
ISMCTS	Information Set Monte Carlo Tree Search
MM	Minimax
PIMC	Perfect Information Monte Carlo
UCT	Upper Confidence Bound 1 applied to Trees
UCB1	Upper Confidence Bound 1
AI	Artificial Intelligence
NAST	N-gram Average Sampling Technique
EPIC	Episodic Information Capture and reuse
EBF	Effective Branching Factor

Chapter 1

Introduction

This chapter serves as an introduction to the problem and scope of this dissertation. Section 1.1 details the context in which the problem is relevant. Section 1.2 properly defines the main issues to solve. Section 1.3 explains the reasons that motivate this work and goals that have been set as an end result. Section 1.4 lists the full report structure by specifying the topics of each chapter.

1.1 Context

In recent years, advances in AI research have developed strong adversarial tree search methods, specifically the Monte Carlo Tree Search (MCTS) family of algorithms. These algorithms have proven to be very efficient in combinatorial games such as the board game Go, where previous attempts at artificial intelligence did not produce results capable of challenging human players. Since the official proposal of MCTS in 2006 [Cou06], many enhancements and variations have been studied and experimentated on several game domains, including games of hidden information, where tree search complexity is relatively high and classic algorithms, such as expectimax, can not give the optimal move in a feasible amount of time, due to the large branching factor of hidden information game trees. MCTS variations such as Determinized UCT and Information Set MCTS (ISMCTS) have shown great results in card games such as Spades [WCPR13b], surpassing state of the art rule based systems specifically built with game domain knowledge. Rule based AI systems are difficult to create, requiring a large amount of research and experimentation on specific game rules, while MCTS algorithms are aheuristic, i.e. they do not require specific domain knowledge to work, and thus variations of MCTS that are found to produce good play on certain games can be re-used on games that share similar characteristics, facilitating AI development for programmers and researchers.

1.2 Problem Definition

MCTS has proven to be efficient in many different game domains, but the most basic implementation, UCT, usually yields simple and rudimentary playing strategies that are incapable of challenging strong human players. To improve the playing strengh of MCTS, there are several algorithm variations and enhancements that can be of use, improving performance on specific domains.

The problem of selecting which variations are best applied to a specific game domain, in order to achieve champion status, is still very challenging. Most search dynamics are not yet fully understood and thus, there is currently no better way than a manual experimentation and analysis of enhancement performance on each specific game [BPW+12].

Games of hidden information add complexity to the tree search due to an increased branching factor, since there is a greater amount of possible states, corresponding to all different combinations of what the hidden information might reveal to be. The final decision process must then be weighted according to the probabilities associated with each possible game state. The common approach to address this issue is through what we define as a determinization: the process of transforming a stochastic game of hidden information into a deterministic one. Afterwards, we can apply algorithms that have been extensively researched for deterministic games, such as MCTS or Minimax, averaging the best move over many determinized games. However, these approaches are ineffective in some game domains due to adverse issues of determinization, such as strategy fusion and non-locality. Also, bluffing strategies are difficult to develop if hidden information is removed from a game.

In light of the previous issues, a few questions are now proposed:

Q1. Which MCTS variations and enhancements are best applied to trick taking card games?

Q2. Can an enhanced MCTS develop strong play against traditional AI techniques in trick taking card games?

Q3. Do the advantages of determinization outweigh its shortcomings when applied to trick taking card games? Can the shortcomings be efficiently diminished through specific enhancements?

All three questions require analysis and research on state of the art MCTS enhancements. With the study of successful MCTS implementations in different trick taking games (e.g.: Spades and Hearts), it's possible to gather a list of most promising algorithm variations and enhancements, as well as conclusions regarding game characteristics that most influence algorithm performance. This process will then facilitate testing and experimentation in specific domain of games studied in the scope of this dissertation.

1.3 Motivation and Goals

The motivation behind this dissertation starts with the application of MCTS variants to card games that are not explored in scientific literature, such as Sueca, that is very popular in portuguese speaking countries.

There is also a lack of open source initiatives that promote code re-use of MCTS algorithms and allow developers to implement and experiment with their prefered game domains. A platform that enables easy integration of different AI agents and testing their relative efficiency would benefit the community of game developers and AI researchers alike. Such initiatives can accelerate the discovery of new MCTS enhancements and allow fine tuning of variations applied to specific domains, speeding up experimental research for several games.

Knowledge of the best kind of MCTS implementation applied to a specific game context can lead to a profitable commercial use of an AI for mobile device games. As an example, a MCTS variation has been applied to the card game Spades, producing strategies mostly perceived as strong and intelligent, leading to very positive application reviews and a large amount of downloads [WCPR13b].

Taking into account the motivation, the main goals regarding the scope of this dissertation can be elaborated as follows:

G1. Elaborate a study on the best known MCTS implementations applied in the context of trick taking card games;

G2. Create a framework with some MCTS variations and enhancements, with application in card games;

G3. Develop a framework that enables different AI agents to compete against each other and evaluate relative performance and quality of play;

G4. Use the developed framework to experiment and analyse the best performing MCTS enhancements in a selection of three trick taking card games.

1.4 Report Structure

This report is composed by 7 main chapters.

The first chapter gives an introduction to the problem and scope of this dissertation, detailing the context in which the problem is relevant, how to properly define the main issues to solve, what are the reasons that motivate this work and goals that have been set as an end result.

The second chapter offers a literature review of scientific research that is relevant to the problem at hand, including an analysis of Monte Carlo Tree Search algorithms and enhancements, along with some required background, namely Game Theory, Decision Theory and the multi armed bandit problem. Other subsections detail what algorithms can be applied in adversarial search for imperfect information games.

The third chapter details the main characteristics of trick taking card games and introduces the rules for all three proposed card games, as well as some insight on common playing strategies.

The fourth chapter presents implemented work throughout the dissertation. Specifically, a new determinization algorithm is proposed, and an overall architecture overview is given on both the Botwars framework and the developed MCTS framework for card games.

The fifth chapter describes all algorithms that are used in the experimental results, detailing pseudocode and a quick analysis on their application to the proposed trick taking card games.

The sixth chapter presents the experimental setups and obtained results, with charts and commented analysis of the observed efficiency for all tested algorithms and enhancements.

The seventh and last chapter gives a conclusion to the developed work, starting with an overview of the achieved goals and then an overall reflection on observed results, while proposing guidelines for possible future work.

Chapter 2

Related Work

This chapter introduces the main topics associated with artificial intelligence for trick taking card games. Section 2.1 and 2.2 introduce the concepts of Decision Theory and Game Theory respectively, defining some important formalizations that are the foundation for artificial intelligence in games. Section 2.3 explains the multi armed bandit problem, which is the underlying theory behind Monte Carlo Tree Search methods. Section 2.4 introduces MCTS with a quick overview of how the algorithm works, with subsections describing the most relevant properties of the algorithm, how UCT balances exploration with exploitation and what enhancements can be applied in the context of card games. Section 2.5 includes a list of algorithms that can be applied to games of imperfect information, such as card games.

2.1 Decision Theory

Decision theory mixes probability theory with utility theory, providing a complete and formal framework for decisions made under uncertainty [RN10, Ch.13]. Problems with an utility value defined by decision sequences are pursued in operations research and the study of Markov decision processes.

A *Markov Decision Process* (MDP) models sequential decision problems in enviroments with total observability, through the use of four components: [RN10, Ch.17]

- S: a set of states, where s_0 is the initial state;
- A: a set of possible actions to apply
- *T(s,a,s')*: a transition model that gives the probability of reaching state s' if action a is applied in state s
- *R(s)*: a reward function

Decisions are then modelled as sequences of $(state, action)$ pairs, where the next state is given by a probability distribution, taking into account a given state-action pair. A policy determines which action will be choosen from a given state in S and the aim is to find the policy π that yields the highest expected reward.

If the state is not fully observable, then a *Partially Observable Markov Decision Process* (POMDP) model is required, where an extra component $O(s,o)$ is an observation model used to give the probability of perceiving observation o in the state s.

2.2 Game Theory

Game theory combines situations where multiple agents interact with the concepts of decision theory. A game can be defined as a set of rules that allow interaction between a number $n > 0$ of players to produce specified outcomes. The following components can describe a specific game:

- S: a set of states, where s_0 is the initial state;
- $S_T \subseteq S$: the set of terminal states;
- $n \in \mathbb{N}$: the number of players;
- A: a set of possible actions to apply;
- $f : S \times A \rightarrow S$: the state transition function;
- $R : S \rightarrow \mathbb{R}^n$: the utility function;
- $p : S \rightarrow (0, 1, ..., n)$: the player that will act in each state

A game starts in the initial state s_0 and advances over time $t \in \mathbb{N}$ until a terminal state in S_T is reached. Any given player k_i selects and performs an action (i.e. a move in the game) that causes the transistion of state s_t to the next state s_{t+1} through the application of function f. Every player receives a reward value, which is the *game-theoretic* value of a terminal state, obtained through the utility function R, according to the player performance in the game. The values are specific to different games, but usually a notation of +1, 0 or -1 can be used to respectively represent wins, draws and losses. A player's strategy, i.e. policy, determines the selection probability of a given action a to apply in state s. A *Nash Equilibrium* is reached when all players combine their strategies in a way that no individual player will benefit from unilaterally changing their own strategy [RN10, Ch.17]. A Nash Equilibrium always exists in a game, but computing it is generally considered an intractable problem for games with a large set of states.

Games can be classified by various properties, some of which are [RN10, Ch.2.4]:

- *Zero-sum*: if the summed game reward for all players totals zero (e.g.: two player games where a player wins and another loses);

- *Information*: if the game state is fully visible to all players, or only partially observable (e.g.: card games where players hide their cards from opponents);
- *Determinism*: if there are factors of chance affecting players strategy, causing uncertainty over rewards (e.g: board games with dice rolls determining the result of actions);
- *Sequential*: if actions are sequential or simultaneous (e.g.: whether all players act at the same time or in their own turn);
- *Discrete*: if actions and effects are well determined and defined or with an infinite set of actions and unpredictable results, i.e. continuous.

In this dissertation, the main focus will be toward card games of imperfect information (the game state is partially observable by players) and non deterministic nature (card deck is shuffled at the start of each game), where actions always occur sequentially (each player puts a card face up on the table in their own turn) and in a discrete manner.

2.3 Multi Armed Bandits

In the multi-armed bandits problem, a gambler must choose which bandit machine to play, taking into account that each machine has arms with different probabilities of giving away a prize. The objective of this problem is to maximize the player gain, and this can be achieved by spending the biggest amount of time in the machine with highest reward probability, but the player must also spend time evaluating which machine actually has the highest probability. This is a specific example of the exploitation versus exploration dilemma: one needs to balance exploitation of actions that are currently believed to be optimal with the exploration of other apparently sub-optimal actions that may turn out to be superior in the future. The problem amounts to selecting which bandit arm should be attempted next, taking as an input all the rewards received on all previous trials.

When playing a sub-optimal arm, it is possible to measure the incurred loss between the current received reward and the expected reward from the optimal arm. This measure is called regret, and the regret acumulated on all previous trials of the player is called the cumulative regret. Multi-armed bandit policies, also known as bandit algorithms, can have their strength evaluated by the expected value of the cumulative regret after a specified number n of trials. Auer et al propose *upper confidence bound* (UCB) policies, of which the simplest policy, UCB1, has an expected logarithmic growth of cumulative regret uniformly over n number of trials [ACbF02]. In MCTS algorithms, the move selection problem can be interpreted as a multi-armed bandit problem, where each move is considered a different arm, with unknown reward probabilities. As such, research in the domain of bandit problems can be effectively reused in the context of Monte Carlo Tree Search algorithms.

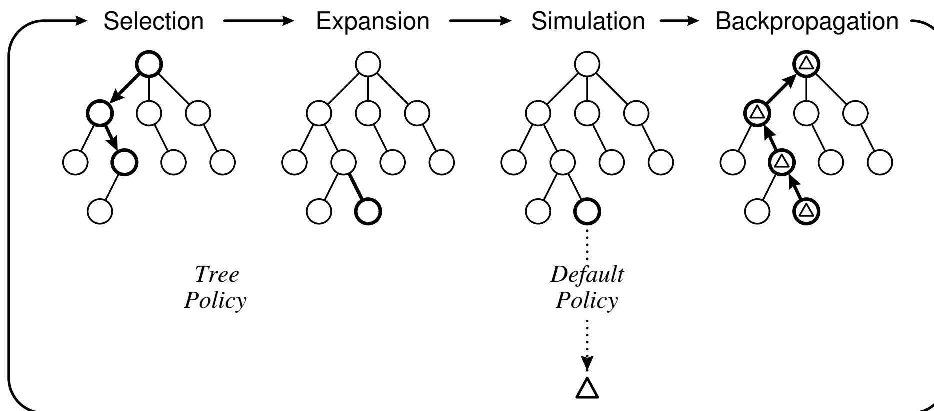


Figure 2.1: One iteration of the general MCTS approach [BPW+12]

2.4 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a family of tree search algorithms that was proposed in 2006 by Rémi Coulom [Cou06]. However, the idea of using Monte Carlo simulations to estimate the game-theoretic value of moves was not new at the time as, for example, it was shown by Abramson in 1991 "to be precise, accurate, easily estimable, efficiently calculable, and domain-independent" [Abr91]. The key innovative aspect of the MCTS algorithm was to combine the use of random simulations for estimating the value of moves with a tree search that guides these simulations toward exploiting promising moves while also exploring untried moves, through the use of the UCT algorithm [Cou06]. MCTS significantly improved the performance of artificial intelligence implementations for various games when other approaches, such as Minimax, were considered inneficient, namely the board game "Go" [GW06].

MCTS is mostly used in games of perfect information, where the UCT algorithm guarantees convergence to the minimax solution [KSW06], but it can also be applied to games of imperfect information through the usage of techniques such as determinization, where good results have been achieved in the card games Skat [BLFS09] and Dou Di Zhu [PWC11].

The most basic MCTS algorithm starts by building a search tree until a pre-established computational limit (e.g.: a fixed number of iterations) is reached. At this point, the best performing root move is returned. Each node represents a game state and the links between states represent actions leading to subsequent states. Although there are many variations of the MCTS algorithm [BPW+12], all of them follow an identical set of four steps in each iteration [CBSS08] (represented in figure 2.1):

1. *Selection*: From the root node, a child selection policy is recursively applied to descend the tree, until the most urgent and expandable node is found. The selection policy usually balances the exploitation of good nodes with the exploration of untried nodes. A node is considered expandable if it represents a non-terminal state and has unexpanded children;

2. *Expansion*: add a child node to the tree, according to available actions from the selected node (i.e. expand a child node);

3. *Simulation*: run a simulation from the newly added node until the end of the game, according to a simulation policy, and return the achieved result;

4. *Backpropagation*: the result is backpropagated through each parent node, in order to update statistics (such as node visit count and node total reward) up until the root node.

Precisely how each step is carried out varies between MCTS algorithms, but the major components that are subject to change are the selection policy and simulation policy. For example, in the UCT algorithm [KSW06], the selection policy is UCB1 and simulations are carried out with completely random moves.

2.4.1 Properties of MCTS

MCTS has several relevant properties when compared to other adversarial search algorithms. It can be applied to any problem modelled as a decision tree without the use of specific domain knowledge other than the game rules. This is possible due to reward statistics, gathered from Monte Carlo simulations, that give an estimate on the actual value of a game state. As such, MCTS can be labelled as *aheuristic*, since other alternatives such as Minimax require specific heuristics to evaluate the quality of game states for the player, which greatly determines the algorithm overall quality of play. For example, in Chess, there are efficient and reliable heuristics that have been developed over years of research, but in the case of the board game Go, where branching factors are much higher and where there are no efficient ways to evaluate the quality of different game states, Minimax performance is simply not enough to challenge good players.

MCTS is an *anytime* algorithm, as it can be stopped at any moment during it's execution, always returning the best found solution at that point. In comparison, Minimax must fully complete the search down to a certain depth of the tree in order to make a decision. By using iterative deepening, the process can be improved to stop at any time, but the best decision still falls back to the last fully explored level of depth in the tree and, as such, the decision only improves when an entire level of depth is fully explored, which can take a great amount of time due to the exponential increase of the number of nodes. MCTS decision progress improves in a relatively smaller granularity, since the outcome of each game is backpropagated immediately throughout the tree, ensuring all values are up to date on each iteration of the algorithm.

MCTS explores the decision tree in an *asymmetric* manner, instead of fully exploring all nodes until a certain depth of the tree. This behaviour is due to the balancing of the exploitation of more promising nodes with the exploration of less visited nodes. Less amount of time is spent visiting parts of tree with sub-optimal moves, in detriment of searching plausible future lines of play in greater depth. This behaviour is visible in figure 2.2

There are also known weaknesses of the MCTS algorithms. It has been demonstrated that minimax is superior in search domains where good heuristics have been researched [RSS11].

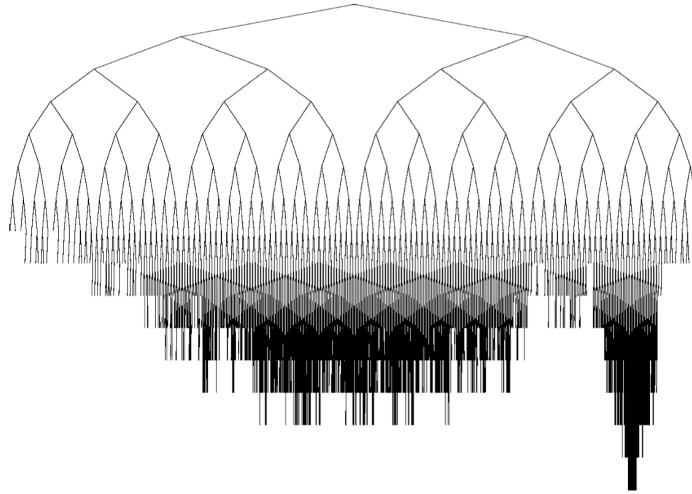


Figure 2.2: MCTS assymetric tree growth [CM07]

In most implementations of MCTS, the integration of enhancements and domain knowledge is required to achieve good results and challenge champion players [BPW+12], but this integration does not necessarily lead to an improved playing strength, since some enhancements can make no meaningful impact or can even be detrimental to the effectiveness of the algorithm [GS07].

> "There is currently no better way than a manual, empirical study of the effect of enhancements to obtain acceptable performance in a particular domain. A primary weakness of MCTS, shared by most search heuristics, is that the dynamics of search are not yet fully understood, and the impact of decisions concerning parameter settings and enhancements to basic algorithms are hard to predict. Work to date shows promise, with basic MCTS algorithms proving tractable to "in the limit" analysis. The simplicity of the approach, and effectiveness of the tools of probability theory in analysis of MCTS, show promise that in the future we might have a better theoretical understanding of the performance of MCTS, given a realistic number of iterations." [BPW+12]

2.4.2 Upper Confidence Bounds for Trees (UCT)

The success of MCTS, particularly in domains such as Go, is mostly due to the *Upper Confidence Bound for Trees* (UCT) tree search proposed by Kocsis and Szepesvári [KSW06]. This is the most popular algorithm in the MCTS family and it combines UCB1 as a selection policy with random play as a simulation policy. In this algorithm, the selection of a child node is treated as a multi-armed bandit problem, with the value of each child being the expected reward estimated by Monte Carlo simulations, and as such, these rewards are analogous to random variables with unknown distributions in the multi-armed bandit problem.

UCB1 has important properties: it is proven to be within a constant factor of the best possible limit on the growth of regret [ACbF02], which makes it a great candidate for the exploitation-exploration dilemma in MCTS. In the selection phase, a child node i is choosen to maximise the following value:

$$UCT = \overline{X}_i + C\sqrt{\frac{\ln n}{n_i}} \tag{2.1}$$

where n is the number of times the parent node has been visited so far, n_i the number of times child i has been visited, C is a constant (higher than 0), and $\overline{X}_i$ is the average reward obtained after n_i simulations from child i.

The first term of the equation focuses on the exploitation and the second term on exploration. As node visits start to increase, the denominator on the exploration term starts to increase, lessening its contribution. If sibling nodes are visited instead, then the numerator increases and thus the exploration of unvisited nodes is promoted. The exploration term ensures that every child has some probability of being selected, which is required due to the random nature of the simulations. If given enough time, even low reward children are guaranteed to be selected, allowing for different lines of play to be thoroughly explored.

The constant C determines the weight of the exploration component in the formula. If it is a high factor, then selection of unvisited nodes is favoured. Based on probability theory, the proposed value of $C = 1/\sqrt{2}$ is proven to satisfy Hoeffding's inequality, which provides an upper bound on the probability that the sum of random variables deviates from its expected value [Hoe63]. However, this is only true for rewards between the range $[0,1]$ [KSW06]. If rewards are outside this range or algorithm enhancements are applied, some experimentation may be required to discover the best observable value.

2.4.3 Enhancements

There are numerous enhancements available for MCTS algorithms [BPW+12], and most of them can be grouped in two different classes, depending on the use of specific domain knowledge. General purpose enhancements can improve performance without using any specific domain knowledge and can potentially bring benefit to any MCTS implementation, with the cost of added complexity in the selection or simulation policy. The added complexity must be carefully evaluated since it can significantly reduce the number of simulations run in a given amount of time. Even though general purpose enhancements are applicable in any implementation, some are only effective in certain game domains that have specific characteristics. As an example, the AMAF enhancement works best in games where the notion of a move being classified as good or bad is independent of the current state of the game [GS11]. Enhancements that take advantage of domain knowledge are well studied in many domains, such as the use of patterns in Go [GS11], but these enhancements aren't the main focus in this dissertation, where experimentation is concentrated on a type of card games, and not only a specific game of the genre.

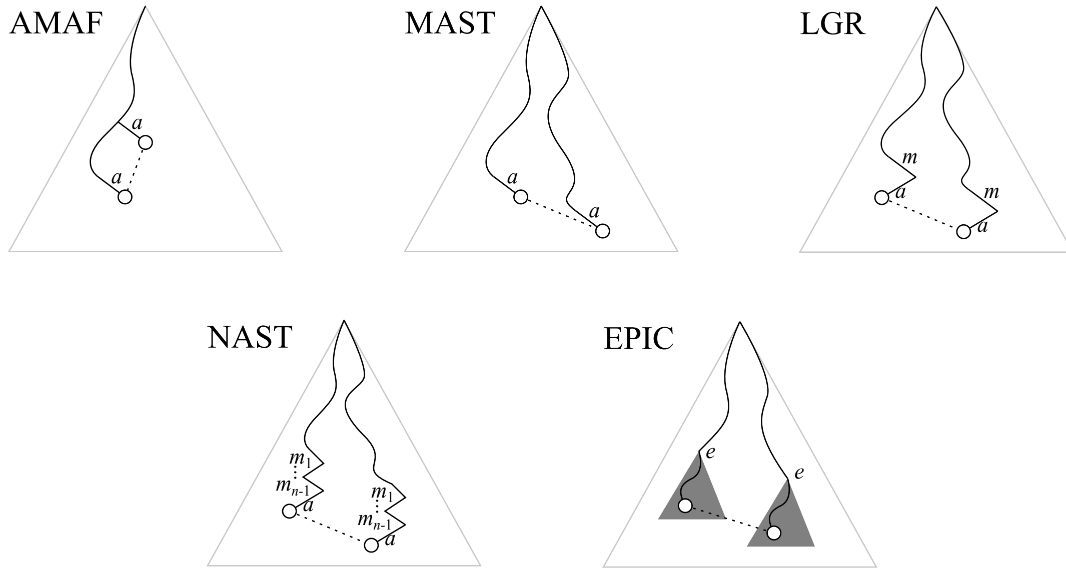


Figure 2.3: A representation of node correlation for various MCTS enhancements. In each case, the strategy is based on the notion that values of the nodes connected by a dotted line are correlated, and thus sharing information between those nodes is beneficial. [PCW14].

Some relevant general purpose enhancements are detailed as follows (with visual representation in figure 2.3):

- *All moves as first* (AMAF) is a technique first proposed by Brügmann as an improvement of Monte Carlo methods for Go [Bru93], and was first mixed with MCTS by Gelly and Silver [GS07]. The principle behind this enhancement is that the value of a move is independent of the point in time when it is played. Games with pieces that rarely move, such as Go, show a significant boost in performance when AMAF is applied. [GS11]. After a simulation is performed on a node, the statistics of that node are updated as well other nodes that represent earlier moments in the same line of play where the exact action that led to the original node could be legally played.

- *Move-average sampling technique* (MAST) was first proposed by Finnsson and Bjöornsson [FB08] and is similar to AMAF, but in this case the average reward statistics are shared for an action independently of where that action occurs in the game tree, and these statistics are then used to influence the simulation policy. This implies that the value of a move is completely independent of game state.

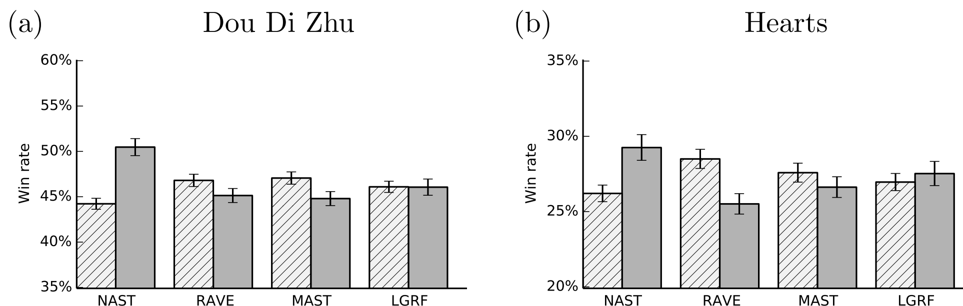


Figure 2.4: Average playing strength results for different combinations of MCTS enhancements tested for Dou Di Zhu (chinese trick taking card game) and Hearts. In each pair of bars, the left-hand bar is the average win rate over the eight combinations not featuring the enhancement in question, and the right-hand bar the average over eight combinations that do feature the enhancement [PCW14].

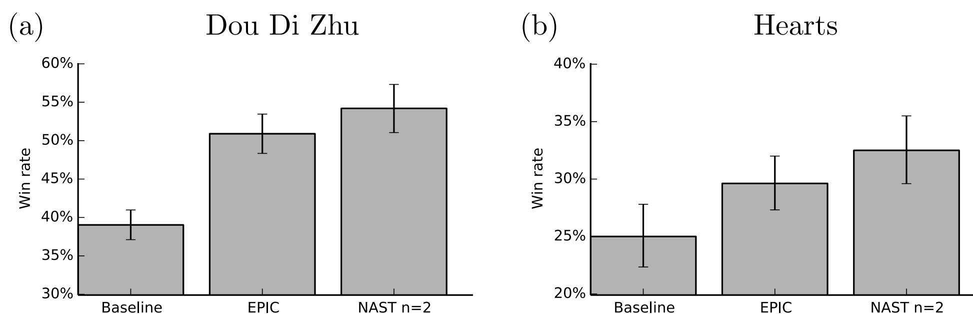


Figure 2.5: Playing strengths (against baseline UCT player) for Dou Di Zhu (chinese trick taking card game) and Hearts, with an agent using EPIC for simulations, and an agent using NAST with n-gram length 2 over 1000 trials each. [PCW14]

- *Last good reply* (LGR) is a simulation policy proposed by Drake [Dra09] where each action is seen as a reply to every opponent's previous move. When a player wins a game, LGR records his replies and then uses them during simulations: if a good reply was recorded, it is used instead of random play. This improvement has shown good performance when applied to Go [Dra09].

- *Episodic information capture and reuse* (EPIC) was proposed by Powley et al [PCW14] and introduces the concept of grouping sequence of moves into episodes, such as the rounds in a trick taking card game. Information is shared between nodes that correpond to the same position in different episodes, allowing to exploit correlations between values of nodes in different subtrees of the whole tree. This means that during the selection policy, average rewards for each node are analysed taking into account statistics from other nodes in the same episode. This enhancement is shown to improve win rate in card games Dou Di Zhu and Hearts [PCW14].

- *N-gram average sampling technique* (NAST) is a generalisation of the MAST enhancement, proposed by Powley et al [PWC13], where instead of learning values only for a single move, NAST learns values for sequences of consecutive moves in the game, with N equal to the length of the sequence. This enhancement is shown to improve win rate in card games Dou Di Zhu and Hearts [PCW14], with results visible in figures 2.4 and 2.5.

Powley et al proposed a framework for describing and combining MCTS enhancements called *Information Capture And ReUse Strategy (ICARUS)* [PCW14], allowing to better understand existing enhancements, mixing them in the same implementation and designing new enhancements. It's thus possible to combine all of the previously described enhancements into a single MCTS implementation, with policies that use a weighted decision between all the enhancements' statistics. In experiments, it is shown that EPIC and NAST increase the win rate of MCTS implementations in the card games Dou Di Zhu and Hearts, while RAVE, MAST and LGR are mostly detrimental [PCW14], with results visible in figures 2.4 and 2.5.

2.5 Adversarial Search in Imperfect Information Games

This section is focused on algorithms specially created to handle games of uncertainty and hidden information. Subsection 2.5.1 introduces determinization techniques with PIMC, with subsections detailing drawbacks of the approach and how to determine when a game has characteristics that benefit from this method. Subsection 2.5.2 introduces minimax and its most important variation for games with chance events, expectimax. Subsection 2.5.3 introduces ISMCTS, which is another variation of MCTS that also uses determinization, but overcomes some drawbacks of *Perfect Information Monte Carlo* (PIMC), with a subsection detailing parallelization enhancements.

2.5.1 Perfect Information Monte Carlo

According to game theory, in games of imperfect information, states are combined into an information set when some player has a perspective of the game state that another player does not. Using card games as an example, players hide their cards from opponents and as such, the information set accounts for states that represent all the possible combinations of opponent cards. Thus, the player will aim to maximize their expected reward taking into account all the possible game states, since it is impossible to distinguish in which specific state he is in.

Determinization, also know as Perfect Information Monte Carlo (PIMC) is a possible approach for handling stochastic and imperfect information games [LSBF10]. In such games, a determinization is an instance of the equivalent game of perfect information where all chance events are known in advance by choosing a current state of the game from the observer's possible information set. These determinizations can be sampled multiple times from a game state and in each determinized version of the game, AI algorithms for perfect information games can be applied, such as MCTS or Minimax. The overall best decision is achieved by combining the best found moves for each determinized game.

Successful uses of PIMC include Ginsberg's GIB program, which plays the card game Bridge at the level of human experts [Gin01]. GIB starts by sampling possible set of cards D, that are consistent with the observed state of the game so far, taking into account game rules. For every deal of cards $d \in D$ and for every move $m \in M$, in which M is the candidate set of possible moves, the perfect information game is searched with a highly optimised exhaustive search to evaluate the score $s(m,d)$, which is the result of applying move m with the deal d. The last step involves choosing the move m which maximizes $\sum_{d \in D} s(m,d)$.

Bjarnason et al propose a UCT algorithm variation to handle stochastic games, named *Sparse UCT*, and applied it to the single-player card game Klondike Solitaire [BFT09]. Bjarnason et al also proposed an ensemble variant of the same algorithm, where several search trees are explored independent from each other and statistics for every move at root nodes are averaged to find the best overall move, with one specific variant of this technique called *HOP-UCT*. In *Sparse UCT* and variants of the algorithm, the game is handled as perfect information game, but instead of determining all possible chance events at the start of the game, these are only determined in the point where information is about to be revealed (e.g.: when face down cards are turned over). This approach works well in single player games where hidden information does not influence the game until it is revealed, but this is generally not the case in multiplayer card games, where hidden information greatly influences style of play right from the beginning of the game and not only when cards are played by opponents.

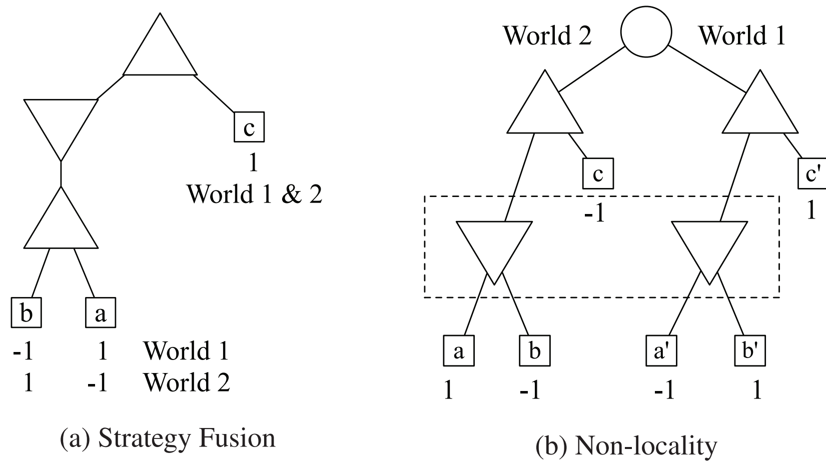
2.5.1.1 Drawbacks of Determinization

Determinization approaches have proven to be very successful in card games such as Bridge [Gin01], Klondike Solitaire [BFT09] and the trick taking card game Skat [FB13]. However, the approach has some flaws that hinder its usefulness in certain game domains. Russel and Norvig describe PIMC as "averaging over clairvoyance" [RN10] and point out determinization will never make an information gathering play nor an information hiding play (i.e. it will never force opponents to reveal their information nor try to hide information from them), since the game is treated as a perfect information game, where all possible moves are visible to all players. Effectively, the algorithm is hindered in terms of bluffing capabilities.

Frank and Basin have identified two other key issues with determinization: [FB98]

- *Strategy Fusion*: arises because PIMC incorrectly assumes it is possible to use a different strategy for differents states in the same information set. However, a player can not distinguish between states in his information set, and must choose the same strategy in each situation.

  An example situation where this is highly detrimental to the algorithm is described as follows in figure 2.6 (a): at the root of the game tree, the maximizing player can choose a move to the right, leading to the terminal state c with a reward of 1. If the option on the left is choosen, the player can receive a reward of 1 or -1 depending on the world he is in and what second choice is made, leading to terminal nodes a or b. An analogous situation



(a) Strategy Fusion

(b) Non-locality

Figure 2.6: Examples of strategy fusion and non-locality. △ and ▽ respectively denote the maximizing and minimizing player decision nodes. Squares represent terminal game states with respective reward. ◯ represent chance nodes. [LSBF10]

would be the maximizing player guessing which side of the coin would be up after a coin toss. For PIMC, both choices would seem equal and thus randomly picked, since it can not distinguish the expected reward between nodes a, b and c, because determinizations lead the algorithm to believe it can always choose the best node in every situation (the coin toss is simulated before the choice is carried out). This is a very poor decision since always choosing the end state c would give a guaranteed positive reward of 1, while the left branch only leads to an expected average reward of 0 (if the coin toss yields a 50% chance of success).

- *Non-locality*: occurs when some determinizations lead to very unlikely situations of play due to opponents' ability to direct play away from these corresponding states, which renders solutions irrelevant to the final decision process.

  An example situation where this effect is prejudicial to the algorithm is visually demonstrated in figure 2.6 (b): there is a chance node at the root of the tree. The maximizing player knows the chance action, and can distinguish if he is located in world 1 or 2. In world 1, the maximizing player has the ability to win the game instantly by choosing terminal state c' or he can let the adversary play. In world 2, the player loses if he chooses terminal state c and thus should let the adversary play. An analogous situation would be in a card game, where the maximizing player could have a game winning card or instead a losing one, as well as cards that would resume the play. PIMC can not distinguish which world the minimizing player is in between states displayed in the dotted rectangle, when in reality it is possible to infer that if the maximizing player had the game winning card, he would rationally make the decision to immediately win, and thus the only plausible state where the minimizing player would have an opportunity to play would be in world 2. Any

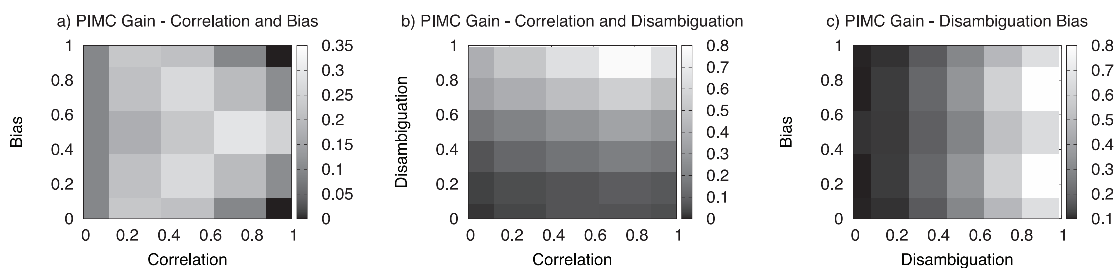


Figure 2.7: Performance gain of PIMC search over random against a Nash equilibrium. Darker regions indicate minimal performance gain for using PIMC search over random play. Disambiguation is fixed at 0.3, bias at 0.5 and correlation at 0.75 in figures a, b and c respectively [LSBF10]

simulations carried out on world 1 for that unlikely state would not be useful for the overall decision and strategy.

2.5.1.2 Measuring Effectiveness of Determinization in Games

Long et al identified three distinct parameters of game trees characteristics and analysed the correlation between the parameters and the effectiveness of determinization [LSBF10]. The parameters are as follows:

- *Leaf Correlation*: the probability of all sibling terminal nodes have equal reward value. Low correlation is present in games where a player can always affect their payoff even very late in the game.
- *Bias*: the probability that the game will favor a specific player over another.
- *Disambiguation factor*: determines how quickly the number of nodes in a player's information set shrinks with regard to the depth of the tree, i.e. how quickly hidden information is revealed as the game progresses. In trick taking card games, a player reveals a card on every move, which means the number of states in each players information set is reduced as the game is being played, leading to a high disambiguation factor. Conversely, in poker, no information is revealed until the end of the round, which translates into a low disambiguation factor.

By performing experiments on synthetic game trees, generated to satisfy different values for each of the three parameters, Long et al have compared the effectiveness of PIMC against random play, with results shown in figure 2.7 [LSBF10]. PIMC search is at its best in games with a high disambiguation factor coupled with a high leaf correlation, which suggests that trick taking card games like Hearts and Skat are ideal candidates for a successful application of determinization techniques, while games such as Poker will most likely show very weak results.

2.5.2 Minimax

The minimax algorithm [RN10, Chapter 6] is a popular tree search method for combinatorial games and has been successfully applied in many domains of perfect information games, producing for example the first program capable of beating a chess grandmaster [Hsu04]. There are variations of the minimax algorithm that handle games of stochastic nature, starting with the expectimax search, where values of chance nodes correspond to the value of child nodes multiplied by the probabilities of the chance outcome. Other variations include for example $*$-minimax trees [Bal83], which also handles chance events and miximax search, and is similar to a single player expectimax [BDS+04].

Unfortunately, expectimax and its' variations are not well suited for trick taking card games, where there is a single chance node at the root of the tree with a very high branching factor, that corresponds to all possible combinations of opponent distribution cards. As an example, in the card game Sueca, there are 40 initially shuffled cards, of which every player receives 10. This means that there are $\binom{30}{10} * \binom{20}{10} * \binom{10}{10} = 5\ 550\ 996\ 791\ 340$ possible combinations of opponent hands in the start of the game.

2.5.3 Information Set MCTS

Whitehouse et al studied the performance of deterministic game search algorithms, namely a cheating minimax (that could observe opponents cards), a determinized minimax (PIMC) and expectimax in a simplified version of Dou Di Zhu. [WPC11] In their experiments, they find that expectimax outperforms determinized minimax by around a 30% higher win rate, while only 10% lower win rate against a cheating minimax, which leads the authors to conclude that the benefit of cheating algorithms, i.e. with the ability to see adversaries' cards, has less to do with having access to hidden information and more with overcoming the weaknesses of determinization, at least for the case of a simpler version of Dou Di Zhu.

When applying PIMC, strategy fusion arises since the value of moves is estimated according to specific determinized states. To overcome this effect, Cowling et al propose a new variation of MCTS named *Information Set MCTS* (ISMCTS) [CPW12], where only a single decision tree is used, in which nodes correspond to information sets instead of game states. Since searching a tree with nodes for each unique information set is intractable for large games, ISMCTS groups information sets into single nodes as much as possible to reduce tree complexity. Also, on each iteration of the algorithm, only one determinization is generated to avoid biasing the value of nodes for specific states.

The main differences between ISMCTS and PIMC, besides the removal of strategy fusion, are the use of a single decision tree instead of multiple trees (comparable in figures 2.8 and 2.9), resulting in lower memory use, and the removal of the need to balance determinizations and simulations, since a PIMC coupled with MCTS requires the specification of the number of iterations per each generated determinization, while ISMCTS uses only one determinization per iteration. However, ISMCTS is more complex compared to a UCT search and thus the amount of simulations done in

the same time frame is significantly reduced. This fact can make PIMC more efficient in specific game domains, were the advantage of more simulations outweigh the effects of strategy fusion. Both algorithms do not address the issue of non-locality (inference) and bluffing, each requiring additional specific enhancements to the algorithm.

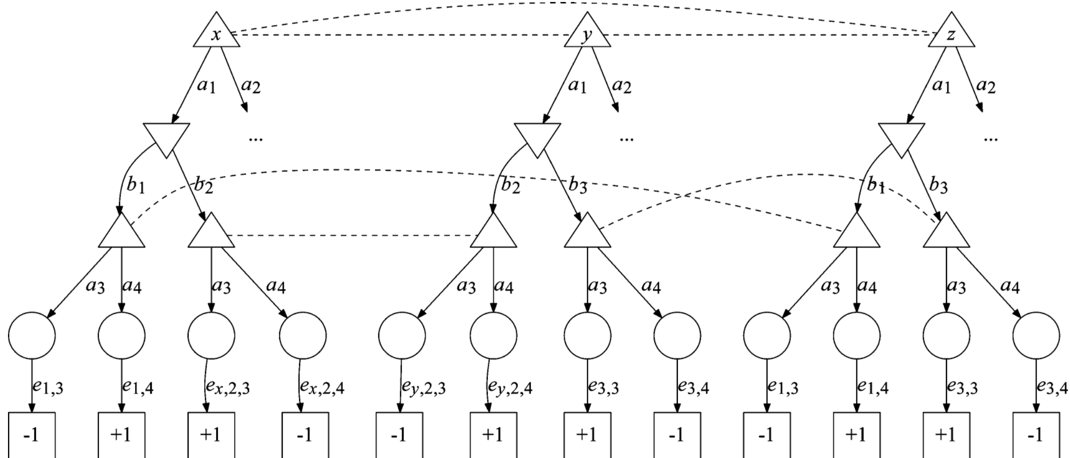


Figure 2.8: An example game tree for a simple 2-player game. Nodes shaped △ denote player 1 decision states, ▽ player 2 decision states, ◯ environment states, and terminal states labelled with reward values for player 1 (the game is zero-sum, so player 2's rewards are the negation of those for player 1). Player 1's information set relation is shown by dashed lines for selected nodes. The partitioning of the remaining nodes is determined by their positions in sub-trees: if two nodes occupy the same position in two sub-trees, and the roots of those sub-trees are in the same information set as each other, then the two nodes are in the same information set as each other. the remaining nodes are partitioned in the obvious way. Player 2 has perfect information, i.e. his information sets are singletons. [CPW12]

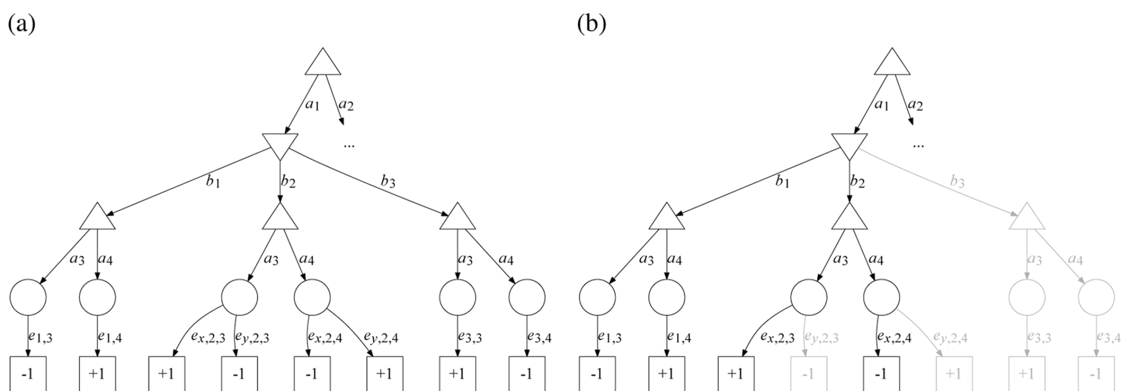


Figure 2.9: An information set search tree for the game shown in (a) shows the entire tree; (b) shows the restriction of the tree to determinization x. [CPW12]

ISMCTS works similar to pure UCT implementation: every node includes children for all possible moves at that game state, according to the available information sets. However, at each

iteration of the algorithm, a determinization is generated and the selection policy is then restricted to nodes that are coherent with the determinized state (visible in figure 2.9). Each node stores the number of times it was available during the selection policy, and UCB1 selection formula is modified to take into account the node availability count:

$$\overline{X}_i + C\sqrt{\frac{\ln n'_i}{n_i}} \tag{2.2}$$

where n'_i is the number of times the child node was available after determinization, n_i the number of times child i has been visited, C is a constant (higher than 0), and $\overline{X}_i$ is the average reward obtained after n_i simulations from child i. Effectively, n, the number of times the parent node was visited is simply replaced by the child availability count. However, this means the selection of the most urgent child node is no longer identical to the multi-armed bandit problem: it is actually a variation where only a subset of the arms are available on each trial, named a *subset-armed bandit*. Whitehouse pointed out that the subset-armed bandit problem introduces some theorical issues with ISMCTS: the value of a move for an opponent may be different for each information set [Whi14, Chapter 5.2.1]. In cases with large number of information sets, it is assumed that the average value accross all opponent information sets can be used to measure the utility of an opponent move, but this leads to the inverse problem of strategy fusion, *strategy fission*, where the opponent is assumed not to be able to choose different actions depending on which information set they are in.

Cowling et al experimented with the ISMCTS algorithm in different hidden information games, specifically the board game Lord of the Rings: The Confrontation (LOTR:C), Phantom (4, 4, 4) game and the chinese card game Dou Di Zhu [CPW12]. It is shown that ISMCTS significantly outperforms PIMC only in LOTR:C, and consistenly performs slightly worse than PIMC in the other domains. The authors conjecture that this discrepancy is due to the fact that ISMCTS can perform a deeper search in the same computational budget, since it utilizes a single tree. They conclude that domains where deep search is possible and beneficial or where strategy fusion is detrimental, ISCMTS will most likely offer a better performance. In domains where information sets have large numbers of legal moves and where the effect of strategy fusion is not clear (such as Dou Di Zhu), ISMCTS does not offer any immediate benefit over other determinization approaches.

2.5.3.1 Parallelization

ISMCTS and other variants of MCTS, such as UCT can be enhanced to distribute processing power across multi-threading hardware environments, speeding up the overall decision process. However, there are multiple proposed enhancements to approach parallelization of MCTS, of which some are detailed as follows:

- *Root Parallelization*: multiple threads run MCTS independently from the same game state but with different random seeds, and the results for each root node are combined to determine the overall most visited node [CJ07];
- *Tree Parallelization*: a single tree is shared between threads, that add nodes and update node statistics. Thread safety is maintained using concurrency mechanisms to avoid writing at the same time in the same memory and also to avoid reading altered memory [CWH08];
- *Tree Parallelization with Virtual Loss*: a variation of Tree Parallelization to discourage selection of nodes that are currently locked by adding losses in the node statistics, reducing the time spent waiting for a lock;
- *Leaf Parallelization*: a single tree is searched by a single master thread that orders worker threads to run simulations when a child node is selected.

Sephton et al have experimented the use of these 4 enhancements in the strategic card game Lords Of War, both using PIMC and ISMCTS [SCPW12], and found that root parallelization is the most time efficient approach, but improvements were only visible up to 4 to 5 threads. Tree parallelization is less effective when applied to ISMCTS, possibly due to the large branching factor of the tree. The reasoning is that the root node has a bigger number of children (since ISMCTS has nodes for all possible moves given the information set) and thus more time is spent waiting for synchronization in the initial phase of the algorithm.

Chapter 3

Trick Taking Card Games

In this chapter, an introduction is made for all three proposed card games that are used in the experimental results. For each game, the rules are thoroughly explained and a quick analysis is done on possible playing strategies, for a better understanding of game mechanics.

3.1 Main Characteristics

A simple standard deck of 52 cards allows to play a multitude of different games. However, many dificulties arise when attempting to classify card games [Par90, p.61-64]. A possible perspective for classification is to group games by their mechanism, i.e. what action is carried out when a player turn is up. This kind of classification can be useful when experimenting with adversarial search algorithms, since games with similar mechanics and branching factors can possibly benefit from the same algorithms and enhancements. As such, trick taking games consist of a category in mechanism classification, where each player in turn plays one card face up on the table, composing a trick. Many other subgroups can be descendants of this category, such as Point Trick Games, where the score won from a trick depends on each individual card value, and players attempt to maximize points won through collected tricks [Pag16b]. Most card games share two important characteristics: they are non-deterministic and of imperfect information, since the deck of cards is usually shuffled at the beginning of each game and players hide their cards from each other.

In artificial intelligence research for games of imperfect information, good results have been achieved in trick taking card games such as Hearts [Stu08], Spades [WCPR13b], Skat [FB13] and Bridge [FB98]. However, many of these games feature bidding phases or simultaneous moves (such as passing cards and then performing a trick in Hearts). These situations add greater complexity and require specific algorithm enhancements and variations. For simplicity, only the simplest form of trick taking card games will be analysed in this dissertation, where players can only complete the current trick, without bidding or taking other possible lines of action, as is the case of the card game Sueca.

3.2 Sueca

Sueca is a popular point-trick game in portuguese speaking countries, specifically Portugal, Brazil and Angola [Pag16d]. The game has four players in teams of two against two, with partners sitting opposite to each other. From the standard deck of 52 cards, the 8s, 9s and 10s are removed, totalling 40 cards in the start of the game. Values of each card are as follows:

Card Rank	Ace	7	King	Jack	Queen	6,5,4,3,2
Points	11	10	4	3	2	0

Before the game starts, a randomly chosen player shuffles the deck of cards. Imagining a table composed of players A,B,C,D, where A and C are partners facing against B and D. Assuming A is the shuffler, player C must split the shuffled deck into two stacks and join them back in any order (this is also known as cutting the deck) and give the resulting deck for player D to distribute. Player D must choose to reveal an unknown card from the top or bottom of the deck, for him to keep. The revealed card's suit becomes the trump suit for that game. If the card was selected from the top, he must distribute 9 cards for himself, and then 10 cards to each player on his left. If he chooses the bottom, he starts by distributing 10 cards to each player to his right, ending with 9 cards for himself. Player A then initiates the game be playing the first card.

Each player chooses a card to play sequentially in a counter-clockwise order, forming a complete trick of 4 cards. Players must use cards that follow the same suit chosen by the trick leading player. If a player does not have cards of the leading suit, then any card can by played. If any cards of the trump suit are in a trick, the highest trump wins. Otherwise, the trick is won by the highest card of the leading suit. The winner of a trick leads the next trick, and his team scores the points equivalent to the sum of card values in the won trick.

The objective is to win tricks containing more than half the card points. A game starts with the possibility of collecting a total of 120 points. The team that gathered 61 or more points wins 1 game point. However, if the team scores 91 or more, it counts as 2 game points. If the team takes all the 120 points, then it counts as 4 game points. Ties do not yield game points for each team. According to official rules of Sueca tournaments held in Portugal [dBT16], the winning team is the first to score 10 game points.

The strategies for playing this game are based on counting cards and then analysing chances of the rival team having certain cards and suits. The game is supposedly played without any comunication between players, since any cue as to what cards each other has will affect the overall playing strategy. A team intends to win high valued tricks, but if they play valuable cards of a suit that the adversary does not have, then a trump can be used and the trick is lost. As such, it is important to get rid of suits of which high valued cards are still in the game, creating opportunities to trump. Most of the focus is toward winning aces and sevens since they are valued as 11 and 10 respectively, while face cards total 9 points on each suit, and numbered cards do not have any value. Some luck is involved in the gameplay, because if a team collectively starts with most aces and sevens, it will be difficult for the adversaries to trump those cards (unless a very unbalaced

distribution of suits is in play). Expert players usually use signals and visual cues to indicate their team mates some information on the suits they have, minimizing the risk of playing certain high value cards.

Many moves also indicate a lot about what cards players hold: if a high valued card is thrown into a losing trick, it is a likely indication that the player used the last card of the playing suit. This is a clear tell sign that the other team should not play more cards of said suit, since they now risk being trumped by the losing player. Another possibility is that the player has higher valued cards of that suit in his possesion, and threw the least valuable one. Interpreting the situation heavily dependends on what cards are still unknown, since we can only assume that the losing player did not have cards of said suit with lesser value to throw into the losing trick.

3.3 Bisca

Bisca is a trick taking game that is very similar to Sueca. Internationally this card game is called Briscola and is particularly popular in Italy, but also played in other countries such as Slovenia and Croatia [Pag16a]. Briscola can be played in variants of two, three, four, five or six players. The two player variant is somewhat popular in Portugal and is by some considered the traditional Bisca, which has small rule differences compared to the international Briscola, but still shares mostly identical gameplay.

Bisca pits two players against each other. From the standard deck of 52 cards, the 8s, 9s and 10s are removed, totalling 40 cards in the start of the game. Values of each card are as follows:

Card Rank	Ace	7	King	Jack	Queen	6,5,4,3,2
Points	11	10	4	3	2	0

Before the game starts, one player shuffles the deck, and deals 9 cards to each player. This number can be variable (in Briscola it is usually 3 cards, but when playing with only two players, a common Bisca variation is to start with 9 cards, for a more strategic play). The final card of the deck is revealed (remaining in the end of the deck) and it will be the trump suit until the game finishes. The other player (non-dealer) starts the game by playing the first card. The remaining card deck is kept aside with initally 22 cards (40 cards minus 9 cards for each player).

A trick in a two player Bisca game is composed of two cards. The leading player can use a card of any suit to initiate the trick. The second player then has a few choices:

- If he follows the same suit with a card that has higher value than the leading card, he wins the trick;
- If he plays another suit, he did not follow the leading suit and loses the trick;
- If he uses a card of the trump suit, the highest trump card wins the trick (note that if the leading suit is trumps, a higher trump must be played to win the trick).

Every time a trick is finished, each player takes a card from the deck, with the trick winning player taking the first card from the top of the deck. Note that when the deck only has two remaining cards, the player who lost the trick will keep the last deck card, which is the trump card that was visible at the start of the game. The player that wins the trick, leads the next one.

When the deck has no remaining cards, the rules change: the leading suit of a trick must now be followed if possible. In other words, before this point any player could opt not to follow the leading suit and discard any card as he wished, but now all players are forced to play only cards that are of the leading suit, if they have any. Only when a player does not have cards of the leading suit, can he choose to play a trump, or throw away cards of any other suit.

After a game is finished, the player that scored most points from his won tricks wins the game. A total of 120 points are available in the beginning, and if a player scores above 61, he won the game.

This game is similar to Sueca, but has some major key differences in strategy. A player does not know beforehand what cards will be available for him to play throughout the whole game. In the first phase, one can choose to discard inconvenient cards, which allows to build a useful hand in the end game. This makes up the key strategy: a leading player tends to avoid using high value cards, because the second player would likely trump (he is not forced to follow the suit). Thus, the second player has a good opportunity to score points, or to discard suits for which he has a low number of cards. In the second phase of the game, when the deck is empty, the leading player will likely play high value cards, because the second player must follow his suit (which is a strategy similar to the one in Sueca).

Reaching the end game phase with lots of trumps and a very low number of different suits with high value cards is critical: the other player would lead with a suit that will probably be trumped, or overthrown with higher valued card. To reach this situation, it makes sense to sacrifice some point cards in the initial stage, if they belong to a suit that has a scarce number of cards in our hand and if there are still valuable cards of that suit in play. As such, the game mixes luck with skill, because of handling the current hand while continuously drawing unknown cards. If players keep count of cards in previous tricks and think some moves ahead in the game, they will likely build a strong hand that will give them the advantage in the last stage of the game.

3.4 Hearts

Hearts is a worldwide popular trick taking card game. The game has many versions in different countries, but we will focus on the American variation and rules. Contrary to the previously described trick taking games, Hearts can be classified as a reverse game since the objective is to avoid collecting card points [Pag16c]. Other differences include players not being grouped into teams, although situations may occur where helping another player may be mutually beneficial. There is also no trump suit. The full standard deck of 52 cards is used with the following value scale (from highest to lowest rank): Ace, King, Jack, Queen, 10, 8, 9, 7, 6, 5, 4, 3, 2. Card points are as follows:

Card Rank	Any Hearts Card (♡)	Queen of Spades (Q♠)
Points	1	13

The objective is to score the least points possible. A game ends when one player reaches 100 points, granting a single victory to the adversary with least score at that point. However, since this definition makes a full game last a highly variable number of moves, we simplify the notion of a complete game to a span of only 52 moves, i.e., when all players have no cards left (this is often referred to as a hand, and we consider that a game is composed of only one hand).

Thirteen cards are dealt to all four players. The first move does not involve playing a card, instead each player must pass 3 chosen cards to another adversary. In the first hand, players give cards to the adversary on their left, in the second hand it is to their right, and in the third hand it for the player in their opposite direction (two players to their left or right). In the fourth hand there is no card passing. Passing cards introduces the concept of partially observable moves into Hearts, which could require specific experimentation with algorithm variations that handle such features. As such, to effectively test the same algorithms in identical conditions for Sueca, Bisca and Hearts, without specific adaptations just for Hearts, we simplify the rules definition and consider that there is no card passing in each Hearts game, making it the case that all games are effectively played as the fourth hand. However, integrating the full rule set of card passing would still be trivial and would not impede the correct functioning of the proposed search algorithms, but it will be considered as a future improvement for this specific game.

The player that holds the two of clubs (2♣) must initiate the first trick with this card. In a clockwise sequential order, each player must then complete the trick by playing cards that follow the leading suit. If players do not possess cards of the required suit, they can choose whichever card they hold. The player with the highest card of the leading suit in the trick scores points equivalent to the trick value and then initiates the next trick.

It is illegal to lead a trick with hearts (♡) before this suit has been drawn. This means that hearts can only lead after at least one player used a card of hearts in a previous trick with a different leading suit (this action is known as breaking hearts). Exception to this rule takes place when a player only holds hearts in his hand.

If some player wins a trick that contains any hearts card or the queen of spades, they collect points. However, if one single player collects all hearts and the queen of spades, then the score is reversed: that player scores 0 points and every other adversary receives 26 points, the worst possible score. This strategy is known as shooting the moon and is considered to be a very risky approach, but experts players will usually attempt it if the odds are favourable, which can yield a very high reward.

Playing strategies share some small similarities to those in Sueca, but in reverse: players attempt to discard very high ranked cards in the beginning, which lowers the chances of winning tricks when hearts start to be played (i.e., when players start running out of specific suits). In the beginning, players also attempt to use low ranked spades, forcing a player to use the queen of spades (this is also known as flushing out the queen). After the initial rounds where aces and other high cards are in drawn, players will try to get rid of scarce suits in the current hand and

draw hearts as fast as possible. However, it is also very important to focus on safely discarding the remaining highest ranked cards. By keeping track of all used cards, players can infer to a certain degree what others might use if a specific card is chosen, but even the best of moves is greatly influenced by luck, as the reward is determined by what others have in hand.

Chapter 4

Implementation

In this chapter, the main focus is toward the developed by-products that were essential to achieve the end results of this dissertation. Specifically, a new determinization algorithm is proposed, which is the foundation for all determinized MCTS algorithms applied in the experimental section. Also, the two frameworks used to carry out experimentation are explained in detail.

4.1 Determinization Algorithms

In this chapter, the following terminology is used:

- An **Assignment** or **Proposal** is a concrete solution of a determinization problem. In the case for card games where players hands are concealed, there is a multitude of assignments that make up all possible hands that the other players might have.

- A **Sample** is a list of assignments consecutively generated by a sampling algorithm. It is important to have a generator that creates uniform samples, i.e. all possible assignments have an equal probability of being generated.

To use the determinization algorithms described in section 2.5.1 and 2.5.3 in trick taking card games, we must be capable of sampling opponent card assignments where every unique assignment has an equal chance of being generated. For this purpose, a simple rejection sampling technique can be used, such as the Accept-Reject algorithm [CRW04], where a proposal is drawn from the set of all possible opponent card permutations and then tested to check if it matches a valid distribution according to game rules. For the specific case of Sueca, Bisca and Hearts, players may not have certain suits if they did not play a card of the leading suit in a previous trick.

However, the Accept-Reject algorithm has an inherent issue of needing an unknown amount of rejections to generate a valid proposal. In situations where restrictions are minimal, this can result in a very low average of rejections, which is usually the case for the three card games that were experimented with. At the start of each game, there are no restrictions, but players will eventually

run out of certain suits, and near the end, despite having a low number of cards, the algorithm will likely spend more time generating invalid assignments, since many restrictions are set (e.g.: player 1 does not have spades, player 2 does not have diamonds, and so forth).

These delay issues with determinization can be tolerable, but there are certain scenarios that can occur in the beginning of a game which completely halt the search algorithm: it is possible to shuffle the deck in such a way that the distribution of suits is very unbalaced. A simple scenario with four players would be as follows:

- Player 1 is the current player and can lead the trick with a card of any suit;
- Player 2 does not have spades but can have any other suit;
- Player 3 does not have spades but can have any other suit;
- Player 4 can have any suit;
- Each player has 8 cards, and in Player 1's perspective, there are 6 unknown cards of each suit (spades, diamonds, hearts and clubs).

To generate a valid assignment, we must shuffle an array of 24 possible unknown cards. The first 8 cards of the array go to Player 2, the next 8 go to Player 3 and the last 8 cards go to Player 4. To create a valid assignment folowing this premise, the last 8 cards can not have a single spade. In short, there are ${}^{24}P_8 *{}^{16} P_8 *{}^8 P_8 \approxeq 6,204 \times 10^{23}$ possible permutations, of which only ${}^{18}P_8 *{}^{10} P_8 *{}^8 P_8 \approxeq 1,291 \times 10^{20}$ are valid. Using a uniformly distributed random number generator, this yields a success rate of about $\frac{1}{4807}$, which means that on average we will require 4807 attempts to reach a single valid assignment. Also, we must bear in mind that an algorithm such as ISMCTS must generate a assignment for each iteration, and in the experiments we use 10 000 iterations as a standard. For this specific use case, it would require on average 48 070 000 array shuffles and validity checks.

As such, a new approach is required to quickly generate valid card assignments that make up a uniform sample from the set of all possible assignments, i.e. not biasing the generation of specific card assignments. As such, an approach is proposed to handle the generation of valid card assignments, without exploring the invalid search space.

4.1.0.1 Randomize non-deterministic game states in card games

In the beginning of a Sueca game, the playing deck usually starts with 40 unique cards, and 4 players.

The following notation is used:

- X_i is a vector of cards that represents the player i hand. Cards contained in this vector may be known or unknown from the current player's perspective;

Let the following be true:

- $C = K \cup U$, in the perspective of a single player, the total set of cards C is composed by the subgroup of cards K (known cards) and U (unkown cards). Every card $c \in U$ is held in the hand of the other players and is concealed from the player perspective.

- $Poss(X_i, c) = 0 \vee 1$, the possibility of player hand X_i to contain card c can be represented by 0 (impossible) or 1 (possible). In other words this represents if $c \in X_i$.

The objective then is to assign every card in U to all unknown X_i. This is similar in nature to the assignment problem of Operations Research. However, instead of generating a single valid assignment in a deterministic manner, we must sample a variety of possible assignments in an uniform way.

To run the algorithm, an input matrix with the possibilities associated with the problem must be elaborated as follows:

- The rows correspond to X_i, which is the hand of player i

- The columns correspond to a card c to assign

- Each cell represents the value of $Poss(X_i, c)$ (the possibility of $c \in X_i$)

As an example, if we are trying to assign six cards to three different players, where we are Player 1, Player 2 has no restrictions, Player 3 can not have cards of the hearts (♡) nor diamonds (♢) and Player 4 can not have spades (♠), then we have the following possibility matrix:

$$M_1 = \begin{array}{c} \\ X_2 \\ X_3 \\ X_4 \end{array} \begin{array}{c} \begin{array}{cccccc} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

To solve this, we start by summing the columns and rows:

- The sum of a column represents how many players can a specific card be assigned to. If there is a column for which the sum is equal to 1, this means the card can only be held by one player, and should thus be immediately assigned (e.g.: between three players, only one can possibly hold this specific card);

- The sum of a row represents how many possible cards the player can receive, to which we can subtract the amount of cards a player still needs to receive (cards left to assign). If the result is equal to 0, then all cells with value of 1 in that row represent cards that should be immediately assigned to the respective player (e.g.: there are only 2 possible cards that player might hold, and we still need to assign 2 cards to him).

Taking both sums and cards left for each player, we have the following:

$$M_2 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum}-\textbf{CardsLeft} \\ 1 & 1 & 1 & 1 & 1 & 1 & \textbf{6}-\textbf{2} \\ 0 & 0 & 1 & 1 & 1 & 0 & \textbf{3}-\textbf{2} \\ 1 & 1 & 0 & 0 & 0 & 1 & \textbf{3}-\textbf{2} \\ \textbf{2} & \textbf{2} & \textbf{2} & \textbf{2} & \textbf{2} & \textbf{2} & \end{pmatrix}$$

Since there is no $ColSum = 1$ nor $RowSum - CardsLeft = 0$, we must start to assign cards to certain players randomly. However, we must not assign a card that would compromise the restrictions of another player. Using the matrix above as an example: if we began by choosing $A\heartsuit$ and $2\diamondsuit$ for X_2 and then $K\spadesuit$ and $Q\spadesuit$ for X_3, we would reach a dead end: only $7\heartsuit$ and $J\spadesuit$ are left for X_4, but this player can not have spades.

To avoid situations where restrictions are broken, we follow a simple rule of thumb: always start by assigning cards to the most restricted players. With the example matrix above: X_2 needs 2 cards and has 6 possibilities, X_3 and X_4 need 2 cards and have 3 possibilites. This means that X_3 and X_4 are the most restricted players ($RowSum - CardsLeft = 1$, which is the minimum current value). After choosing one of these players randomly, we then have to pick a random card where $P(X_i, c) = 1$.

As an example, by picking X_3: we have the following choices: $[K\spadesuit, Q\spadesuit, J\spadesuit]$. If we would then choose $K\spadesuit$ we would have the following matrix (notice that the $K\spadesuit$ column is now 0 filled and Cards Left for X_3 decreased by 1):

$$M_3 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum}-\textbf{CardsLeft} \\ 1 & 1 & 0 & 1 & 1 & 1 & \textbf{5}-\textbf{2} \\ 0 & 0 & 0 & 1 & 1 & 0 & \textbf{2}-\textbf{1} \\ 1 & 1 & 0 & 0 & 0 & 1 & \textbf{3}-\textbf{2} \\ \textbf{2} & \textbf{2} & \textbf{0} & \textbf{2} & \textbf{2} & \textbf{2} & \end{pmatrix}$$

Applying the same rule as before, we choose between X_3 or X_4. Assuming we assign randomly $Q\spadesuit$ to X_3, we have the next matrix (notice that X_3 is now a 0 filled row: he can no longer hold any other card):

$$M_4 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum}-\textbf{CardsLeft} \\ 1 & 1 & 0 & 0 & 1 & 1 & \textbf{4}-\textbf{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \textbf{0}-\textbf{0} \\ 1 & 1 & 0 & 0 & 0 & 1 & \textbf{3}-\textbf{2} \\ \textbf{2} & \textbf{2} & \textbf{0} & \textbf{0} & \textbf{1} & \textbf{2} & \end{pmatrix}$$

Now we see that $ColSum(J\spadesuit) = 1$. This means that $J\spadesuit$ can only be held by one player, which is X_2, so we must assign it immediately, leading to the following matrix:

$$M_5 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum} - \textbf{CardsLeft} \\ 1 & 1 & 0 & 0 & 0 & 1 & \mathbf{3-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0-0} \\ 1 & 1 & 0 & 0 & 0 & 1 & \mathbf{3-2} \\ \mathbf{2} & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \end{pmatrix}$$

We should now choose player X_4, and we can randomly assign $A\heartsuit$ to him, leading to:

$$M_6 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum} - \textbf{CardsLeft} \\ 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{2-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0-0} \\ 0 & 1 & 0 & 0 & 0 & 1 & \mathbf{2-1} \\ \mathbf{0} & \mathbf{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \end{pmatrix}$$

Now we can choose between X_2 or X_4. Let's assign X_2 with $7\heartsuit$, leading thus to the final assignment of $2\diamondsuit$ to X_4.:

$$M_7 = \begin{array}{r} \\ X_2 \\ X_3 \\ X_4 \\ \textbf{ColSum} \end{array} \begin{pmatrix} A\heartsuit & 7\heartsuit & K\spadesuit & Q\spadesuit & J\spadesuit & 2\diamondsuit & \textbf{RowSum} - \textbf{CardsLeft} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0-0} \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0-0} \\ 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \end{pmatrix}$$

Following these steps guarantees that we will never assign a card that would eventually be required by another player: we always start with the column that has the least possibilities, and if a situation arises where a card can only be held by a single player, then it is automatically assigned.

One can argue that this approach does not generate samples in a randomly uniform way due to certain branching decisions: in M_4 we were forced to choose only one card because of the decision we took before on M_3. If we were to choose a different card in M_3 then another decision branch would occur in M_4. Since we are always starting with the most restricted players, the least restricted ones might only reach certain combinations of cards through multiple previous decisions in the restricted players, making these combinations rarer.

However, we must take into account that the order of assignments is irrelevant, and that the search space can be so vast that 10 000 randomizations would not be enough to bias a distribution space of over 1×10^{20} possibilities. With these aspects in mind, it is possible to assume that even if some bias toward certain combinations occurs, it would be negligible compared to the strenghts of this approach: it is guaranteed that we reach a valid distribution in only one attempt, so we have a clear upper bound on the worst case scenario, making it on average faster than the pure random Accept-Reject approach. Another positive aspect is that it only uses simple matrix operations.

Algorithm 1 Pseudocode of proposed Uniform Assignment Sampling with Restrictions Algorithm

```
 1: function ASSIGNWITHRESTRICTIONS(unknownCards, players)
 2:     possibilityMatrix ← GENERATEPOSSIBILITYMATRIX(unknownCards, players)
 3:     for each index in [0, size of unknownCards]  do
 4:         if the sum of a column in the matrix is equal to 1 then
 5:             select card of the respective column to the only player where the cell value = 1
 6:             ASSIGNCARDTOPLAYER(card, player, possibilityMatrix)
 7:         else
 8:             randomly select a player where (rowSum − playerCardsLeft) is min. and > 0
 9:             choose a random card from the player's respective row, where the cell value = 1
10:             ASSIGNCARDTOPLAYER(card, player, possibilityMatrix)
11:     return players with assigned cards
12:
13: function GENERATEPOSSIBILITYMATRIX(playerHands)
14:     initialize matrix with size (players ∗ unknownCards)
15:     for each card in unknownCards  do
16:         for each player in playerHands  do
17:             select cell in matrix where col = cardIndex and row = playerIndex
18:             if player has unknown cards in hand and might hold card by the game rules then
19:                 set selected cell value to 1
20:             else
21:                 set selected cell value to 0
22:     return matrix
23:
24: function ASSIGNCARDTOPLAYER(card, playerHand, matrix)
25:     decrease playerCardsLeft by 1
26:     zero all elements in the respective card column
27:     if playerCardsLeft = 0 then
28:         zero all elements in the respective player row
29:     return updated matrix
```

Table 4.1: Comparison of samples generated with the proposed generator approach versus accept-reject method, using a Chi Squared test with H_0 = *sample is uniform* and H_1 = *sample is not uniform*.

Distribution	Expected	Accept-Reject	Proposed Approach
$\langle 2\diamondsuit, J\spadesuit\rangle, \langle K\spadesuit, Q\spadesuit\rangle, \langle 7\heartsuit, A\heartsuit\rangle$	1111	1113	1117
$\langle 2\diamondsuit, K\spadesuit\rangle, \langle J\spadesuit, Q\spadesuit\rangle, \langle 7\heartsuit, A\heartsuit\rangle$	1111	1087	1114
$\langle 2\diamondsuit, Q\spadesuit\rangle, \langle J\spadesuit, K\spadesuit\rangle, \langle 7\heartsuit, A\heartsuit\rangle$	1111	1130	1114
$\langle 7\heartsuit, J\spadesuit\rangle, \langle K\spadesuit, Q\spadesuit\rangle, \langle 2\diamondsuit, A\heartsuit\rangle$	1111	1137	1114
$\langle 7\heartsuit, K\spadesuit\rangle, \langle J\spadesuit, Q\spadesuit\rangle, \langle 2\diamondsuit, A\heartsuit\rangle$	1111	1129	1099
$\langle 7\heartsuit, Q\spadesuit\rangle, \langle J\spadesuit, K\spadesuit\rangle, \langle 2\diamondsuit, A\heartsuit\rangle$	1111	1102	1118
$\langle A\heartsuit, J\spadesuit\rangle, \langle K\spadesuit, Q\spadesuit\rangle, \langle 2\diamondsuit, 7\heartsuit\rangle$	1111	1097	1119
$\langle A\heartsuit, K\spadesuit\rangle, \langle J\spadesuit, Q\spadesuit\rangle, \langle 2\diamondsuit, 7\heartsuit\rangle$	1111	1113	1111
$\langle A\heartsuit, Q\spadesuit\rangle, \langle J\spadesuit, K\spadesuit\rangle, \langle 2\diamondsuit, 7\heartsuit\rangle$	1111	1092	1094
χ^2	0	2.3246	0.548
df	8	8	8
p-value	0	0.9694	0.9998
Reject H_0? (*p-value* < 0.05)	no	no	no

A simple experiment can be conducted to analyse a sample that was produced from both random generators and check if it is uniform. Using the specific example matrix presented previously, where the search space is relatively small (only 9 possibilities), we can check if there is any bias directed toward specific assingnments. For these trials, a sample of 10 000 distributions was created using each generator. A Chi squared test was performed, considering that H_0 = *sample is uniform* and H_1 = *sample is not uniform*. The results are visible in Table 4.1 and demonstrate that we can not reject the null hypothesis for both samples, which means that we can not state, with a 95% confidence interval, that both generators did not generate a uniform sample (at the 0.05 significance level).

The previous experiment only verified if a single sample from each generator is not uniform, and it can be the case that those samples were favourable by luck. To verify if the generators are consistent over a large number of samples, we now repeat the previous experiment 1000 times with the same sample size of 10 000 assignments. Results in table 4.2 show that both generators do not create perfectly uniform samples, since we can reject the null hypothesis in about 4.5% of samples for each one. Note that both generators give reasonably the same results, which is the most important aspect to take into account: the new proposed approach seems to have the same behaviour as the accept-reject method. However, this is not actual proof that both generators are uniform, it is a simple experiment to assess if both produce good enough results so that assignment bias does not become a significant issue in the determinization process.

Theoretical proof to justify that the generator is uniform will be considered as future work. However, we will propose some reasoning to support the hypothesis that samples are generated in a uniform way through the proposed algorithm:

1. Firstly, when applying the algorithm, we must assume there is at least one possible valid assignment for a given list of cards and players;

Table 4.2: Uniform Sampling analysis with Chi Squared test for each 1000 samples, with each sample of size 10 000

Approach	Rejected H_0	Rejected H_0	Rejected H_0	Samples Generated
Accept-Reject	44	8	0	1000
Proposed Approach	46	5	0	1000
Significance Level	0.05	0.01	0.001	

2. The order of card assignment does not change the set of possible assignments. With the notation $[X \mapsto N]$, where card X is assigned to player N, from a possible set of cards [A,B,C,D] to assign players [1,2], choosing the assignment order $[A \mapsto 1,\ B \mapsto 1,\ C \mapsto 2,\ D \mapsto 2]$ yields the same result as $[C \mapsto 2,\ D \mapsto 2,\ A \mapsto 1,\ B \mapsto 1]$, since player 1 is assigned with $[A,B]$ and player 2 with $[C,D]$;

3. Given 2., it is possible to choose any order of card assignments to reach the full set of all possible assignments;

4. The proposed algorithm provides an order of card assignments that is guaranteed to always terminate in a number of iterations equal to the count of unknown cards;

5. When the algorithm reaches a non-deterministic assignment decision, it determines the choice using uniform random number generators;

6. By making random uniform decisions regarding the selection of individual card assignments, we achieve an overall general assignment that was uniformly sampled.

4.1.0.2 Biasing randomized game states based on game knowledge

Although it is important to achieve uniform samples of deterministic game states, there are certain use cases where bias can be of use. In a Sueca game, if the player always followed the suit that started the trick, then by the game rules, the player might still hold any specific card of that suit. But we might be perceptive of clues that indicate the player actually does not have that suit. Imagine the following situation in the start of a Sueca game: the trick started with Player 1 pushing an $A♣$. Player 2 played a low card, 2♣, Player 3 played a high card $J♣$. The last player was going to lose the round, but still played a card that gave his adversaries 2 points: the $Q♣$. This gives us two possible assumptions: either Player 4 has cards of clubs (♣) with a higher value than a Queen, or he does not have any card left of that suit (assuming that Player 4 played rationally and is not bluffing). This is actually a lot of information: the remaining cards of clubs above a Queen are $K♣$ and 7♣. If we are Player 1 and hold the 7♣, we can ascertain with a degree of confidence that Player 4 does not hold any clubs other than the King. As such, if we generate samples of assignments that are biased toward Player 4 not having any clubs lower than a Queen, we are selecting a more probable reality according to game rules and rationality, which then significantly narrows the determinized MCTS search space. However, we should not simply reuse the previous algorithms using a possibility equal to 0, since Player 4 can actually be bluffing

or playing irrationaly. As such, we can not be 100% sure of our assumptions, but can have a certain degree of belief.

To support this type of sample bias towards player having certain cards, it is possible to modify the Accept-Reject method. We can increase rejections by adding another validity check: if a card was assigned with low odds of belonging to a player, we can reject the whole assignment with a certain probability. However, this increases the number of rejections, leading to a more delayed and troublesome execution.

However, the proposed sampling algorithm referenced in section 4.1.0.1 can be easily adapted to bias certain card assignments, with little overhead to the algorithm. Instead of generating a possibility matrix, we create a probability matrix, where each cell specifies in a range of $[0,1]$ the belief that a certain player holds a specific card. When summing the values inside a row or column, we must take into account that if a number is higher than 0, it should count as a 1. After selecting a player row, in order to choose the card to assign, we select one of the possible cards based on their odds. For example, a card with 0.8 has higher odds of selection than another with 0.4, but both are still valid options. This means that instead of making random uniform individual decisions (probability equal to all choices), we bias some individual card assignments with a higher probability.

Although this proposed alternative has some potential, specific domain knowledge is required in order to use it: taking into account the previous example, the Sueca implementation needs to provide heuristics that determine odds of each player possessing certain cards. While the study in this dissertation is more toward aheuristic enhancements, this approach might be useful for researchers looking into enhancements tailored for just a specific game.

4.2 Botwars Framework

In this section, an overall description and architecture overview will be given on the framework used to conduct experiments throughout this dissertation.

The Botwars framework was developed by Rui Gonçalves [Gon16] to handle communication between different AI agent implementations, allowing such agents to challenge each other in a myriad of unique games. The framework offers a web app client for game visualization in real time and also enables interaction between human players and the developed AI programs. The framework also serves as an orchestrator for automated competitions between different agents, allowing to set up and monitor a long running number of games, in order to assess the relative efficiency of each agent.

Some open source contributions were done throughout this dissertation to enable database storage support, facilitating the distribution of experimental results in a transparent and verifiable way. Whenever a competition is carried out, all game state changes are stored in a document oriented database (for this specific case, Couchbase). This means that the experimental results are simply database backups that can be shared and restored by any user. This allows for every agent action and game state to be fully queriable through a SQL-like language (N1QL), enabling any

user to aggregate and analyse results with custom perspectives, without requiring experiments to be re-run. After restoring a database, the framework can reload the stored states, allowing for total inspection down to every single move of each played game, through the web client. If agents store the necessary data to replay a move (e.g.: random number generators state), then stored moves can be replayed in a deterministic manner, allowing to inspect data structures created by the algorithms (e.g. search trees) and debug any errors or exceptions.

The framework is not specifically tailored to card games, as any discrete and sequential game can be supported. To implement a new game, a developer must create two different files: the game logic class and the rendering logic class. The game logic class must override specific inherited functions such as:

- *isValidMove*: check if a given move can be applied by the next player in the current game state;
- *move*: apply a given move to the current game and return the resulting new game state;
- *isEnded*: check if game is over;
- *getNextPlayer*: return the next player that must perform an action in the current game;
- *getWinners*: if the game is over, return the players that won the game;
- *getFullState*: return all state variables from the current game state;
- *getStateView*: return variables visible to a specific player (e.g.: hide hands of other players in a card game).

Afterwards, the rendering logic class must be able to receive any given game state (returned by *getStateView*) and draw the game with HTML and CSS (in this specific case, React framework is used to facilitate DOM manipulation).

With both game logic and rendering implemented, the framework then handles the remaining work: it exposes a REST API that agents must communicate with, in order to find and register in hosted games or competitions. After the registration is done, the server opens a websocket connection with each agent and constantly sends events, such as announcing when the game starts, what actions the other players performed, what is the current game state in the perspective of each agent, and also querying each client for the move he wishes to apply (validating everything in the process, to avoid reaching invalid game states). As such, a developed agent program must simply create an interface of communication that sends HTTP requests in order to find and register in a competition, and then use a websocket library for listening and reacting to events sent by the server.

The resulting work gives some advantages to AI developers. Much of the synchronization between agents as well as game setup is automatically handled by the framework, and running experiments simply involves creating scripts to boot the application, backup and clean the database, and then spawn the amount of agents required to play. It allows developers to choose whichever

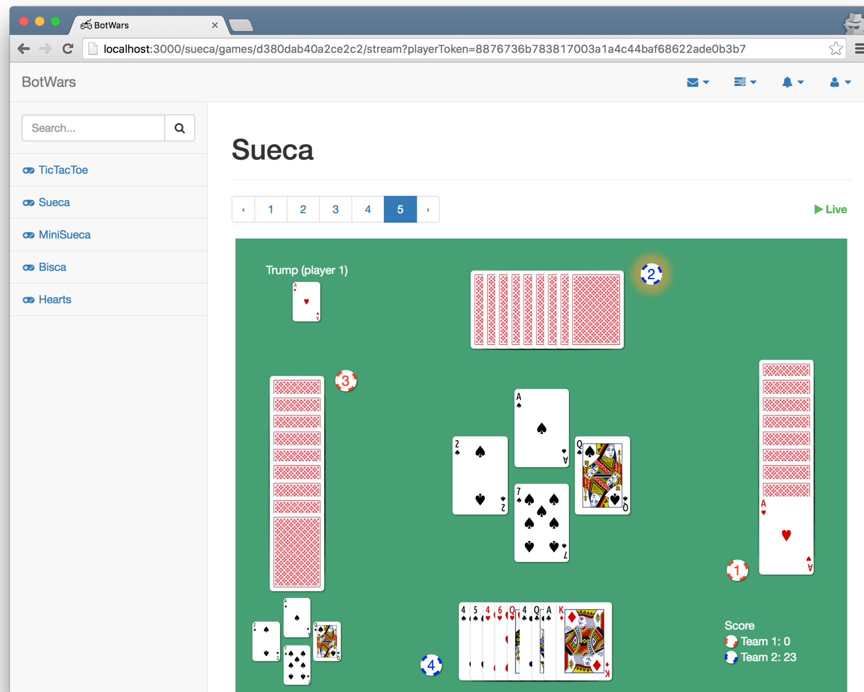


Figure 4.1: The botwars client UI while playing a Sueca game.

programing language they find most adequate to implement their agents, and offers a simple UI for users to play and see competition results.

4.3 MCTS Framework for Trick Taking Card Games

In this section, an overview will be given of the AI agent framework developed specifically for the work carried out in this dissertation [Fer16], that integrates with the Botwars framework described in section 4.2.

The main focus of the MCTS framework is to offer different algorithm implementations and allow runtime modification of core steps through initialization parameters. The framework also facilitates move replayability, enabling deterministic debugging and offering a search tree visualization tool for further inspection on a move decision.

The distribution of source code and datasets generated from all conducted experiments is a critical step towards *Reproducible Research*. This term refers to the idea of publishing papers that are product of academic research along with the full computational environment used to produce the experimental analysis, such as the source code, datasets, scripts and any other required tools [FC09]. This enables other researchers to reproduce the results and create new work based on the existing research. Note that most papers cited throughout this work do not share the necessary

experimental setup to reproduce results. As such, the MCTS framework was built with these aspects in mind: it is an open source project with focus on replayability and code reuse.

When connected to the Botwars server and database, the MCTS framework agents will store all state variables required to deterministically replay any given move. Afterwards, in order to replicate moves, a script can be used to load a game state, instantiate the used search algorithm with the configured parameters, restore the random number generator state and then request the next move. This is also useful for comprehension and enhancement of algorithms, because a developer can spectate a game in search for bad moves, replicate the conditions where a move happened, and then try to understand why that behaviour occured, through inspection of generated search trees or use of a debugger. This enables a quick kind of experimentation and hypothesis testing: what could improve move selection in a given game state? To test multiple theories, a move can be replayed with different parameters: a different random seed, different configurations, different search algorithm, or different type of algorithm enhancements. This was crucial in the implementation of MCTS algorithms, since a lot of erroneous behaviour was quickly detected and fixed through inspection and testing.

Replayability can also be used to quickly test the playing quality of an agent: a suite of tests could be created with some game states where the possible moves are analysed beforehand by human experts. If the algorithm returns poor quality moves, developers can then inspect errors and improve performance before running long term experiments, such as the ones described in chapter 6 (i.e. 1000 games per algorithm variation against a baseline agent).

The framework was built with the three proposed card games, however, note that games are not directly coupled to the search algorithms. This means that new kinds of games can be implemented both in Botwars and the MCTS framework, and the existing algorithms will still work under the assumption that games are discrete, sequential and implement the defined interface functions.

The Decorator design pattern was used to support integration of MCTS enhancements. This pattern allows to add a new behaviour to an individual object, in a static or dinamic way, without modifying the behaviour of other objects from the same class [GHJV94]. When an agent is queried for the next move, a list of enhancements is passed as a configuration for the new instance of the search algorithm class. Each enhancement is a class that should only implement the MCTS functions that require changing, and not rewrite the whole algorithm. The algorithm instance object will then iterate over the list of enhancements, and decorate itself with the behaviour specified in each enhancement class. This enables to mix multiple kinds of enhancements into one instance of a search algorithm, assuming that each enhancement class overrides different functions (i.e. there is no behaviour collision).

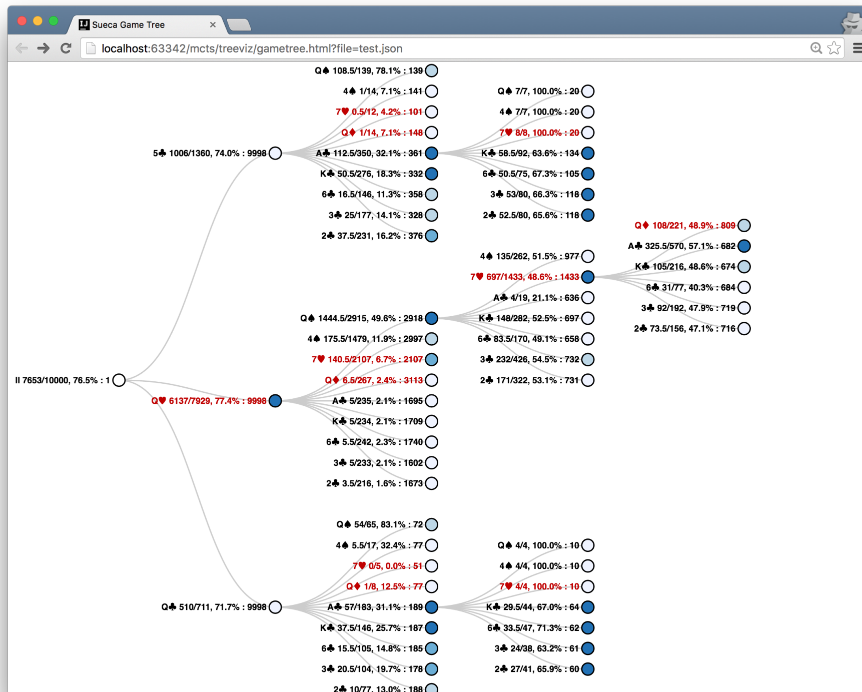


Figure 4.2: The MCTS framework tree visualisation tool displaying a tree generated by ISMCTS during a Sueca game. Each node specifies the move, reward, visits, winning odds and node availability count after determinization. A white to blue color scale is used in each list of child nodes to indicate the most visited siblings.

Chapter 5

MCTS Algorithms applied to Trick Taking Card Games

In this chapter, an overview will be given on all algorithms that are used in the experimental section. Pseudocode and formulas are detailed, along with some effect analysis when such methods are applied to trick taking card games.

5.1 Determinized UCT

Upper Confidence Bound 1 applied to Trees (UCT Search) can not be directly applied to games of non deterministic nature. A quick way to avoid the issue in card games is to sample a random possible card assignment for adversaries hands, and then run the UCT Search algorithm in this version of the game where all player cards are visible.

The pseudocode for standard UCT Search is presented in Algorithm 2 and is the main component in Determinized UCT. The following notation is used in the pseudocode:

- $f(s,a)$ returns the state s' reached by applying action a to state s (state transition function);
- For each node v there is an associated state s_v, the incoming action a_v, the total simulation reward $Q(v)$, the visits count $N(v)$, and the child nodes $c(v)$;
- Δ is the reward vector for a finished game. $\Delta(v)$ is the reward for the player that performed action a_v;
- c is a factor of the standard exploration constant, i.e. with $c = 1.0$, the exploration constant is equal to $1/\sqrt{2}$;
- $A(s)$ = set of possible actions at state s.

For a preset number of determinizations, a card assignment is uniformly sampled from the distribution of all possible assignments. Algorithms that do such sampling are described in section 4.1. After the determinization process, the UCT Search yields the best found move within the given number of iterations. Note that a low iteration count may be sufficient (e.g. around 500) to find the best move since the lack of hidden information leads to a smaller branching factor. When the best move is returned for a search, it counts as a single vote towards that specific move. After all determinized searches have ended, the most voted move is returned.

We will now analyse the tree complexities of each game when applying a single UCT search on a determinized state. Starting with the case of Sueca, the game tree resulting from an exhaustive search, beginning from an initial state of the game, has an upper bound of $(10!)^4 = 1.73 \times 10^{26}$ leaf nodes. This is calculated by considering that, at the game start, each player holds 10 cards and, at the end of the first trick, which is composed of four moves, we can be in one of 10^4 possible states, assuming that all four players can choose any of their 10 cards. In the second trick, we start from one of 10^4 states and for each one we can reach 9^4 new different states, totaling $(10 \times 9)^4$ possible states. Continuing this calculation until all players hold no cards, we get the final result of $(10!)^4$. This is actually an upper bound limit because, for example, there can be at most 10^4 possible outcomes of the first trick, but only the first player truly has 10 choices, while the other three must follow the leading suit, and as such, not all 10 cards might be available for use. Further generalizing the formula, at the end of every trick, we have $(\prod_{t=x}^{10} t)^4$ leaf nodes, where t is the number of cards each player held at the beginning of the trick. When evaluating the subtree representing all possible outcomes of a single trick, we have a total of $(x + x^2 + x^3 + x^4)$ nodes, with each summand representing the number of nodes in a progressing tree depth. Therefore, the total number of nodes for the full tree can be calculated as follows:

$$1 + \sum_{x=1}^{10} \left[\left(\prod_{t=x+1}^{10} t \right)^4 (x + x^2 + x^3 + x^4) \right] = 1.03 \times 10^{27}$$

Where +1 represents the root node. Note that when $x = 10 \Rightarrow t = 11$, leading to $(\prod_{t=11}^{10} t)^4 = 1$. This is intended behaviour for the first trick, since there is only one previous node in the tree, which is the root node. To better understand how this formula develops, we calculate the amount of nodes just for the initial three tricks (with a decreasing x, starting at $x = 10$):

$$1 + (10 + 10^2 + 10^3 + 10^4) + 10^4 \cdot (9 + 9^2 + 9^3 + 9^4) + 10^4 \cdot 9^4 \cdot (8 + 8^2 + 8^3 + 8^4) = 307\,128\,611\,111$$

Each component in parenthesis represents the number of nodes in a full trick, while every summand represents the total number of nodes at each level of tree depth. The first level has 1 node, the second has 10 nodes, the third has 10^2 and so on and so forth.

With this calculation, we now conclude that the leaf nodes represent about 16.75% of the full tree. The total depth of the tree is 40, i.e. the sum of cards each player holds, and therefore, the Effective Branching Factor (EBF) [RN10] is 4.71 using the following calculation (with N as the total number of nodes and d as tree depth):

Table 5.1: Upper bounds on complexity values of Determinized UCT search algorithm applied to Sueca, Hearts and Bisca

Game	State space	Tree nodes	Leaf nodes	Tree depth	EBF
Sueca	4.71×10^{21}	1.03×10^{27}	1.73×10^{26}	40	4.71
Bisca	6.20×10^{36}	5.02×10^{32}	1.30×10^{32}	40	6.54
Hearts	5.36×10^{28}	8.97×10^{39}	1.50×10^{39}	52	5.84

$$N = \sum_{k=1}^{d} EBF^k \implies \sum_{k=1}^{40} EBF^k = 1.03 \times 10^{27} \implies EBF = 4.707391915628215$$

To calculate the state space, the total number of possible card assignments can be given through:

$$\binom{40}{10} \times \binom{30}{10} \times \binom{20}{10} \times \binom{10}{10} \approxeq 4.71 \times 10^{21}$$

This is because there are 40 initial cards and each player consecutively receives 10 cards, where the card order does not matter.

Hearts has exactly the same trick mechanics as Sueca, and as such, only the number of initial cards changes from 10 to 13. All the premises established for calculating number of nodes and leaf nodes are the same. Note that the first move is always the 2♣, and not the full 13 card possibilities, but this makes a negligible difference in the final result.

For the case of Bisca, two players start with 9 cards each and play a card per turn, while also taking a hidden card from the deck. At the 11th turn, when the deck is empty, each player will use a card until no more cards are in hand. Taking these different rules into account, there are $(9^2)^{11} \times (9!)^2 = 1.30 \times 10^{32}$ possible leaf nodes. This means that in the first 11 tricks, each player can use any of the 9 cards they hold, totaling 9^2 outcomes for each trick. After that round, each player consecutively loses a card, and the calculation is equal to the one in Sueca, but only with 2 players instead of 4. To find out the total number of nodes in the tree, we modify the formula used for Sueca:

$$1 + \sum_{n=1}^{2\times 11} 9^n + \sum_{x=1}^{9} \left[9^{2\times 11} \cdot \left(\prod_{t=x+1}^{9} t \right)^2 \cdot (x + x^2) \right] = 5.02 \times 10^{32}$$

Where each summand still represent the amount of nodes in a progressing level of tree depth. But in the initial 22 turns, two players always hold 9 cards each, and as such, the first level has 1 node, the second has 9, the third has 9^2, up to the 11th level with $9^{2\times 11}$ nodes. After this point, each player has a decreasing number of cards, and the formula develops as follows:

$$(1) + (9) + (9^2) + (9^3) + \cdots + (9^{22}) + (9^{22} \times 9) + (9^{22} \times 9^2) + (9^{22} \times 9^2 \times 8) + (9^{22} \times 9^2 \times 8^2) + \ldots$$

To calculate the state space, we must consider the order of cards in the deck. As such, the first player receives 9 cards out of 40, the second gets 9 cards out of 31, and the remaining 22 are in the deck, where the order of cards is important to the gameplay. Thus, we have the following number of possible assignments:

$$\binom{40}{9} \times \binom{31}{9} \times 22! \approxeq 6.20 \times 10^{36}$$

5.2 ISMCTS

Information Set MCTS was proposed by Cowling et al [CPW12], and the algorithm offers an approach that directly handles the issue of stochastic events, while mitigating the effect of strategy fusion. There are different variations of ISMCTS that are suited to support partially observable moves, however, the chosen experimental games do not have these particular moves (note that in Hearts, card switching was simplified), and as such only the standard Single Observer ISMCTS (SO-ISMCTS) is used, which is detailed in algorithm 3. The notation used in the pseudocode is as follows:

- $f(d,a)$ returns the determinization d_0 reached by applying action a to determinization d (state transition function);
- For each node v there is an associated state s_v, the incoming action a_v, the total simulation reward $Q(v)$, the visits count $N(v)$, and the child nodes $c(v)$
- Δ is the reward vector for a finished game; $\Delta(v)$ is the reward for the player that performed action a_v;
- c is a factor of the standard exploration constant, i.e. with $c = 1.0$, the exploration constant is equal to $1/\sqrt{2}$;
- $A(d)$ = set of possible moves in determinization d;
- IS_0 = the current game information set, i.e. the set of all possible states in which the game can be, given the player's observed information;
- $N'(v)$ = availability count for node v;
- $c(v,d) = \{v' \in c(v) : a_{v'} \in A(d)\}$, the children of v compatible with determinization d;
- $u(v,d) = \{a \in A(d) : \nexists v' \in c(v,d) \text{ with } a_{v'} = a\}$, the actions from d for which v does not have children in the current tree. Note that $c(v,d)$ and $u(v,d)$ are defined only for v and d such that d is a determinization of (i.e., a state contained in) the information set to which v corresponds.

Algorithm 2 The UCT Algorithm

```
1: function UCTSEARCH(s_0)
2:     create a root node v_0 with state s_0
3:     while within computation budget do
4:         v_l ← TREEPOLICY(v_0)
5:         Δ ← DEFAULTPOLICY(s_{v_l})
6:         BACKUP(v_l, Δ)
7:     return a(BESTCHILD(v_0, 0))
8:
9: function TREEPOLICY(v)
10:    while v is nonterminal do
11:        if v not fully expanded then
12:            return EXPAND(v)
13:        else
14:            v ← BESTCHILD(v, c)
15:    return v
16:
17: function EXPAND(v)
18:    choose a ∈ untried actions from A(s_v)
19:    add a child v' to v
20:    s_{v'} ← f(s_v, a)
21:    a_{v'} ← a
22:    return v'
23:
24: function BESTCHILD(v, c)
25:    return argmax_{v' ∈ children of v} Q(v')/N(v') + c sqrt(2 ln N(v) / N(v'))
26:
27: function DEFAULTPOLICY(s)
28:    while s is nonterminal do
29:        choose a ∈ A(s) uniformly at random
30:        s ← f(s, a)
31:    return reward for state s
32:
33: function BACKPROPAGATE(v, Δ)
34:    while v is not null do
35:        N(v) ← N(v) + 1
36:        Q(v) ← Q(v) + Δ(v)
37:        v ← parent of v
```

Line 25: $\underset{v' \in \, children \, of \, v}{\arg\max} \frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N(v)}{N(v')}}$

The tree complexity of ISMCTS is significantly higher because using information sets implies consideration of all possible moves at any given point, with the known information at hand, leading to large branching factors. In Determinized UCT, all other player cards were visible, and thus, at most 10 different moves could be played by all adversaries. In the case of ISMCTS, each adversary has the possibility to use any unknown card in game at that point, which means that in the beginning of a game, an adversary can have at most 30 different possible moves.

By analysing tree complexity of ISMCTS when applied to Sueca, there are $10! \cdot 30! = 9.63 \times 10^{38}$ leaf nodes. At the first branch, the root player knows his cards, therefore there are 10 possible branches. At the second branch, the next player can play at most 30 different moves, since there are 30 unknown cards at that point, from the root player perspective. Analogously, the third and fourth branch will have 29 and 28 available moves. The fifth turn belongs to the root player again, now with 9 possible moves. The next branches will have 27, 26 and 25 available moves, and so on until the end. Note that we consider the root player always as the leading player in every trick, but the order of play is irrelevant to the final calculation. To calculate the number of total nodes in the tree, we can use a formula similar to the one in section 5.1, that summed the total number of nodes at each depth level in the tree:

$$1 + \sum_{i=1}^{10} \sum_{j=1}^{4} \prod_{x=i}^{10} x \prod_{y=3(i-1)+j}^{30} y = 2.78 \times 10^{39}$$

Note that when $i = 10 \vee j = 4 \Rightarrow y = 31$, resulting in $\prod_{y=31}^{30} y = 1$. This is intended behaviour for the first level of depth, since the iteration with $i = 10 \vee j = 4$ yields $10 \cdot 1$, which corresponds to the 10 initial possibilities of the root player. To demonstrate how this formula will develop, we calculate the number of nodes in the tree up to the seventh level of depth, with each level in parenthesis (starting with $i = 10$ and $j = 4$, both in a decreasing order):

$$(1) + (10) + (10 \cdot 30) + (10 \cdot 30 \cdot 29) + (10 \cdot 30 \cdot 29 \cdot 28) + (10 \cdot 30 \cdot 29 \cdot 28 \cdot 9) + \\ + (10 \cdot 30 \cdot 29 \cdot 28 \cdot 9 \cdot 27) = 61\ 639\ 811$$

When applying ISMCTS to Sueca, leaf nodes represent 34.64% of the total nodes, with a tree depth of 40, and EBF of 9.66. The state space of the game is not going to differ when compared to Determinized UCT, but we can obtain the Information-Set Space (ISS) by multiplying the number of initial states with the total number of nodes in the ISMCTS tree, thus totalling $2.78 \times 10^{39} \cdot 4.71 \times 10^{21} = 1.31 \times 10^{61}$ information sets.

It is important to note that the calculated values are exact and not upper bounds on the tree size, even considering the rule that players must follow the leading suit if possible. As an example, imagine the first move were the root player leads with an $A\spadesuit$. Considering the current information set, the next player might have spades or not, and as such, any card from the full set of 30 unknown cards can actually be played in that specific moment. Odds might tell that not having spades in the first round is an unlikely event, but it is still a possibility in the current information set.

The calculations for Hearts are the same as Sueca, with the number of cards changing from 10 to 13. However, Bisca needs an adaption for the 11 initial rounds, where two players always

Table 5.2: Complexity values of ISMCTS search algorithm applied to Sueca, Hearts and Bisca.

Game	ISS	Tree nodes	Leaf nodes	Tree depth	EBF
Sueca	1.31×10^{61}	2.78×10^{39}	9.63×10^{38}	40	9.66
Bisca	2.01×10^{73}	3.25×10^{36}	8.39×10^{35}	40	8.15
Hearts	1.97×10^{85}	3.67×10^{56}	1.27×10^{56}	52	12.22

start with 9 cards each. The initial trick outcomes are $9 \cdot 31$, since there are 31 unknown cards. The second trick is $9 \cdot 29$, as one card is revealed from the adversary, and another is taken from the deck. Repeating this until the 11th round, we have $9^{11} \cdot \prod_{x=1}^{11}(9+2x)$ possible outcomes. After this point, each player can deduce what cards the other has, since there are 9 unknown cards, and the adversary holds 9 cards. Both play a card per trick, translating into $(9!)^2$ different outcomes. The final result gives the amount of leaf nodes:

$$9^{11} \cdot \prod_{x=1}^{11}(9+2x) \cdot (9!)^2 = 8.39 \times 10^{35}$$

To calculate the total amount of nodes in the tree until the deck runs out, we must develop the following sum of nodes at each depth level:

$$(1)+(9)+(9 \cdot 31)+(9 \cdot 31 \cdot 9)+(9 \cdot 31 \cdot 9 \cdot 29)+\ldots$$

And the total node count for the full tree can be formulated as:

$$1+\sum_{x=1}^{11} 9^x \cdot [\prod_{n=13-x}^{11}(9+2n)+\prod_{n=12-x}^{11}(9+2n)]+\sum_{x=1}^{9}[(\prod_{t=x+1}^{9} t)^2 \cdot (x+x^2) \cdot (9^{11} \cdot \prod_{n=1}^{11}(9+2n)] = 3.25 \times 10^{36}$$

Where the first summand is the root node, the second corresponds to the 11 first tricks (while the deck is not empty), and the last summand returns the remaining 9 tricks. Note that when $x = 9 \Rightarrow t = 10$, resulting in $\prod_{t=10}^{9} t^2 = 1$, and also when $x = 1 \Rightarrow n = 12$, giving $\prod_{n=12}^{11}(9+2n) = 1$. Both are intended behaviour to calculate the initial move in each phase.

5.3 Reward Functions

For every MCTS algorithm, the reward function is a very important component of the UCT formula, which guides the search toward the most promising moves. By changing the definition of reward achieved by a simulated finished game, we effectively alter what is the search notion of the best move. As such, three different reward functions are proposed as follows:

Algorithm 3 ISMCTS

1: **function** ISMCTS(IS_0, n)
2:     create a single-node tree with root v_0 corresponding to IS_0
3:     **for** n iterations **do**
4:         $d_0 \leftarrow$ randomly choose determinization from IS_0
5:         $(v,d) \leftarrow \textsc{Select}(v_0, d_0)$
6:         **if** $u(v,d) \neq \emptyset$ **then**
7:             $(v,d) \leftarrow \textsc{Expand}(v,d)$
8:         $\Delta \leftarrow \textsc{Simulate}(d)$
9:         $\textsc{Backpropagate}(r,v)$
10:     **return** a_c where $c \in \arg\max_{c \in c(v_0)} N(c)$
11:
12: **function** SELECT(v,d)
13:     **while** d is nonterminal and $u(v,d) = \emptyset$ **do**
14:         **for all** $v' \in c(v,d)$ **do**
15:             $N'(v') \leftarrow N'(v') + 1$
16:         $v \leftarrow \textsc{BestChild}(v,d,c)$
17:         $d \leftarrow f(d, a_v)$
18:     **return** (v,d)
19:
20: **function** BESTCHILD(v,d,c)
21:     **return** $\arg\max_{v' \in c(v,d)} \left(\frac{Q(v')}{N(v')} + c\sqrt{\frac{2 \ln N'(v')}{N(v')}} \right)$
22:
23: **function** EXPAND(v,d)
24:     choose $a \in u(v,d)$ uniformly at random
25:     add a child v' to v
26:     $a_{v'} \leftarrow a$
27:     $d \leftarrow f(d,a)$
28:     **return** (v', d)
29:
30: **function** SIMULATE(d)
31:     **while** d is nonterminal **do**
32:         choose $a \in A(d)$ uniformly at random
33:         $d \leftarrow f(d,a)$
34:     **return** reward for state d
35:
36: **function** BACKPROPAGATE(v, Δ)
37:     **while** v is not null **do**
38:         $N(v) \leftarrow N(v) + 1$
39:         $Q(v) \leftarrow Q(v) + \Delta(v)$
40:         $v \leftarrow$ parent of v

- **Positive Win or Loss (PWL)** is the standard definition of reward used by UCT Search. If a player is the winner of a simulated game, then the reward is equal to 1, otherwise a loss is valued as 0. Ties are counted as 0.5. Note that the UCT formula also takes into account the number of times a node was visited (effectively, the number of simulations that took place after the move associated to the node was selected). This means that the reward component of the UCT formula for a given node will equal to the average win rate achieved by simulations from the node and its children.

- **Win or Loss (WL)** gives negative weight to losses. Simulated game victories count as 1, but losses will incur a penalty of -1, while ties are equal to 0. This may not seem very different between PWL, but the exploitation component of the UCT formula will now be equal to: $\frac{wins-losses}{wins+losses+ties}$. As such, moves that yield more losses than victories will have a negative reward. Moves with a higher win to loss ratio are positively rewarded, but note that the overall reward is lesser in the exploitation component of the UCT formula, compared to PWL. This gives more weight to the exploration component, favouring less tried moves until a significant win to loss ratio is found, and thus an experimental analysis is required to find a proper exploration constant.

- **Score Difference (SD)** is a measure of score difference between the winner and player. For the case of Sueca, wins are ranked as three different outcomes: normal victory (over 60 points equals 1 match point), a significant win (over 90 points equals 2 match points) and total victory (120 points equals 4 match points). We can apply this same victory scale logic to Bisca, since both scoring systems are very similar. However, in order to adapt this reward logic to Hearts, the difference between player and winner score is also discretized into values of [-4, -2, -1, 0, 1, 2, 4], with 4 being the best possible score, corresponding to 0 points in difference (i.e. the player is also the winner), and -4 is the worst score, corresponding to -26 points in difference (i.e. the winner scored 0 while the player has 26 points). Having a score difference higher than -13 points yields a positive reward. The full table of scores difference is shown as follows (each cell contains the value or range of $winner\ score - player\ score$):

Reward	-4	-2	-1	0	+1	+2	+4
Sueca	-120	[-118, -62]	[-60, -2]	0	[+2, +60]	[+62, +118]	+120
Bisca	-120	[-118, -62]	[-60, -2]	0	[+2, +60]	[+62, +118]	+120
Hearts	-26	[-25, -19]	[-18, -13]	-13	[-12, -7]	[-6, -1]	0

Note that this may not be the most adequate reward function for both Bisca and Hearts, since there is no scoring system that values results as 1, 2 or 4. However, if we used the actual score difference instead, we would see maximum score differences of 120 in Bisca and 26 in Hearts, which would require experimentation with very high values of the UCT exploration constant, complicating the experimental analysis setup. As such, having the reward function return discrete values in the range of $[-4,4]$ for all three games allows to test similar exploration constants, and the discretization of scores will not greatly harm the overall perception of reward.

5.4 NAST Simulation Enhancement

The *N-gram average sampling technique* was first proposed by Powley et al [PWC13] and builds upon previous work by Stankiewicz et al [SWU12] and Tak et al [TWB12]. In the context of games, an N-gram is a sequence of N consecutive actions (for the case of card games, N consecutively played cards). The enhancement works by learning a value for each sequence of moves, independent of the context in which they are played. With specific selection policies, the learned sequence evaluations are used to sample more successful moves during the MCTS simulation step.

A sequence of 1 action is simply the move itself and using n-grams of this length effectively is the same as applying the *Move-Average Sampling Technique* (MAST) [FB10], which evaluates the quality of a single move, independent of when it is applied in a game. Using n-grams of $N = 2$ can be thought of as a generalisation of the last good reply principle [Dra09], since evaluating the average reward for any given sequence of actions $\langle a_1, a_2 \rangle$ will indicate if a_2 is a good reply to a_1. In more general terms, the average reward for an n-gram $\langle a_1, \ldots, a_{n-1}, a_n \rangle$ indicates whether action a_n is a good response to $\langle a_1, \ldots, a_{n-1} \rangle$. When applying this concept to trick taking card games, it makes some sense that there is a good reply to any given trick. For example, in Sueca, the sequence of actions $\langle A\spadesuit, 7\spadesuit \rangle$ is generally considered as a bad move, independent of when it is applied, since teams play in alternating turns, and the 7 loses to the Ace. There can be situations where the trick can still be won, but this reply is considered as very risky and should be avoided. However, the sequence $\langle K\spadesuit, 7\spadesuit \rangle$ might not be necessarily good: there can still be an Ace of Spades in play that would overthrow the trick and make the player lose points. If this is the case, then for example $\langle K\spadesuit, 7\spadesuit \rangle$ will likely have negative reward when $\langle 7\spadesuit, A\spadesuit \rangle$ occurs next in the simulation. As such, a reply is choosen by the move sequence, but there is still some notion of context, since later sequences in the simulation will likely affect the reward obtained by previous ones. Note that a sequence can occur in any step of the game, which means that it might be composed of moves from different tricks.

During the backpropagation phase in MCTS, the full sequence of moves from the root node down to last move of the finished simulated game is split into n-gram sequences of N length. For each sequence, the reward function used by MCTS gives the result obtained in the finished game relative to the player that performed the last move in the sequence. This calculated reward is then

stored in a table of sequences for a specific player, where any given sequence corresponds to a number of times it appeared and total obtained reward from all appearances.

During the simulation phase of MCTS, a move is not randomly picked from the list of all possible moves to apply. Instead, each possible move is concatenated with the previous $N-1$ moves that occured in the simulated game, generating a list of possible sequences with N length. A simulation policy determines what is the most promising sequence in the list so far, and applies it in the simulated game. This step is iteratively performed until the simulation ends.

5.4.1 Simulation Policies

There are multiple policies with different criteria for selecting the most adequate move to apply next in any given step of a simulation. These policies are not strictly related to NAST and n-gram sequences, as they can be applied with any set of actions $A(s)$, where each action a has the following information:

- $Q(a)$ = the total reward achieved by applying action a;
- $N(a)$ = the total number of times action a was applied

A policy yields a probability distribution π over $A(s)$. As such, a random uniform number generator must be used to sample an action a with probabilities $\pi(a)$. Note that a MCTS simulation policy should strike a balance between the exploitation of good actions with exploration of other actions, in order to ensure the simulation reward accurately reflects the game theoretic value. With the notation detailed so far, the following policies are described as follows:

- **Gibbs distribution sampling** is the original formulation proposed by MAST [FB08] and defines the simulation by means of a Gibbs distribution:

$$\pi(a) = \frac{\exp\left(\frac{Q(a)}{N(a)}/\tau\right)}{\sum_{b \in A(s)} \exp\left(\frac{Q(b)}{N(b)}/\tau\right)} \tag{5.1}$$

  Where $\tau > 0$ is considered as a factor of greediness because as $\tau \to 0$, the policy will give higher probabilities to actions with maximal reward $\frac{Q(a)}{N(a)}$, and as $\tau \to \infty$, the probabilities all become equal. As such, the value of τ must be tuned through experimentation. Powley et al tested values of $\tau \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1, 1.5, 2, 4\}$, and found $\tau = 1$ gave strongest play in Hearts and Dou Di Zhu [PWC13].

- **ε-greedy** is proposed by Tak et al [TWB12] as an alternative simulation policy for MAST. Let A_{max} be defined as the subset of $A(s)$ for which reward is maximal:

$$A_{max} = \underset{a \in A(s)}{\arg\max} \frac{Q(a)}{N(a)} \tag{5.2}$$

The policy is then defined by:

$$\pi(a) = \begin{cases} \frac{1-\varepsilon}{|A_{max}|} & \text{if } a \in A_{max} \\ \frac{\varepsilon}{|A(s)|-|A_{max}|} & \text{if } a \notin A_{max} \end{cases} \tag{5.3}$$

In this policy, ε is a tunable greedy factor, with values between $[0,1]$. The action of maximal reward is choosen with a probability of $1-\varepsilon$, or any other action with probability of ε. With $\varepsilon = 0$, the policy always chooses the action of maximal reward, while $\varepsilon = 1$ gives an uniform random sampling. Powley et al tested values of $\varepsilon \in \{0,\ 0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.5,\ 0.6\}$ and found $\varepsilon = 0.2$ to be best for Hearts and Dou Di Zhu [PWC13].

- **Roulette Wheel** selects moves with a probability proportional to the average reward of each action. It has no tunable parameters and is defined as:

$$\pi(a) = \frac{\frac{Q(a)}{N(a)}}{\sum_{b \in A(s)} \frac{Q(b)}{N(b)}} \tag{5.4}$$

- **UCB1** can be applied to balance exploitation of high reward actions with the exploration of less explored actions without requiring random sampling. As such, let A_{ucb} be the subset of $A(s)$ for which the UCB1 value is maximal:

$$A_{ucb} = \underset{a \in A(s)}{\arg\max} \left(\frac{Q(a)}{N(a)} + c\sqrt{\frac{\ln \sum_{b \in A(s)} N(b)}{N(a)}} \right) \tag{5.5}$$

The simulation policy is given by:

$$\pi(a) = \begin{cases} \frac{1}{|A_{ucb}|} & \text{if } a \in A_{ucb} \\ 0 & \text{if } a \notin A_{ucb} \end{cases} \tag{5.6}$$

That is, an action that yields maximal UCB1 value is choosen uniformly. By adjusting the exploraction constant c, the search will tend toward moves that are unexplored, and if $c \to \infty$, all actions with less $N(a)$ will be selected uniformly, independent of their reward. Powley et al tested values of $c \in \{0.2,\ 0.5,\ 0.7,\ 1.0\}$ and found $c = 0.7$ to be best for Hearts and Dou Di Zhu [PWC13].

5.5 Minimax

Minimax can not be directly applied to games of imperfect information. However, there are variations such as Expectimax that handle situations of stochastic nature, by introducing the concept of chance nodes at any given point in the search tree. For the case of all three trick taking card games, the only chance event happens in the game start, where the card deck is shuffled. Assuming the deck is shuffled in a uniform way, every possible card distribution has an equal chance of appearing. This means that applying Expectimax at the root node is actually the same as using Determinized Minimax, also known as PIMC coupled with minimax, similar in nature to the Determinized UCT described in section 5.1. All possible card assignments are generated, and a minimax search is run on each version of the game where all player cards visible (note that this way, certain optimizations such as $\alpha\beta$-pruning can be applied). Each search returns the best move for that given game, and counts as a vote for the best overall move. Once all searches end, the most voted move is returned. Note that this strategy suffers the same effects as Determinized UCT, most notably strategy fusion and non-locality, described in section 2.5.1.1. The implemented minimax pseudocode is detailed in Algorithm 4.

It is unfeasible to calculate a minimax search from the beginning of a card game without a proper heuristic function to evaluate any given game state, even if we know other players cards. To further complicate this problem, there is a multitude of possible card assignments that are contained in the current information set, and a minimax search for each possible assignment is necessary to properly evaluate the best overall move. As such, we can only apply minimax search at a certain point near the last move. For the case of Sueca and Hearts, a feasible point of application is at 13 moves before the end. For Bisca, this number increases to 15, but note that at this point the game is deterministic: the adversary can only hold what cards are left in play.

Algorithm 4 The Minimax Algorithm with $\alpha\beta$ pruning

```
1: function ALPHABETA(node, depth, α, β, isMaximizingPlayerTurn)
2:     if depth = 0 or node is a terminal node then
3:         return the heuristic value of node
4:     if isMaximizingPlayerTurn then
5:         v ← −∞
6:         for all child of node do
7:             v ← max(v, ALPHABETA(child, depth − 1, α, β, false))
8:             α ← max(α, v)
9:             if β ≤ α then
10:                break
11:        return v
12:    else
13:        v ← ∞
14:        for all child of node do
15:            v ← min(v, ALPHABETA(child, depth − 1, α, β, true))
16:            β ← min(α, v)
17:            if β ≤ α then
18:                break
19:        return v
20:
21: ALPHABETA(rootNode, depth, −∞, +∞, true)
```

Chapter 6

Experimental Results

In this chapter all the algorithms and enhancements described in chapter 5 will be tested in the developed framework with the three proposed games: Sueca, Bisca and Hearts. It is not feasible to measure an absolute performance of each individual enhancement or algorithm, and as such, a baseline agent is defined for every test. Each variation must compete in 1000 games against the baseline, to measure its relative performance.

The charts in each experiment will focus on two main metrics: score and win rate. All three games have a great emphasis on the final game score. For the case of Sueca, games can have a value of 1,2 or 4 match points depeding on the final score, and in an official Hearts game, the first player to reach 100 points ends the overall match, with the winner as the least scoring player. As such, the definition of victory for each game may not be enough to perceive the overall playing quality of the agent. For Sueca and Bisca, a game is won by scoring more points than the adversary team. In Hearts experiments, we define winners as the players that scored the least amount points in an single game of 52 moves (for example, if two players score 0 points, then they are both considered as winners). But imagine that an experiment for two Hearts games gave the following results:

Player	A	B	C	D
Game 1 Score	26	0	26	26
Game 2 Score	0	5	7	14
Sum of Points	26	5	33	40
Win Rate	50%	50%	0%	0%

The table indicates that Player B scored 21 points less than Player A but both win rates are at 50%. With this information, scores charts for Hearts would appear to have more significance than the win rate charts. However, note that the algorithms used throughout the experiments attempt to maximize a reward function, which awards victory to whom scores the least amount of points in a single game, and not necessarily the least amount of points possible. Taking this into account, both scores and win rates are important for the result analysis.

Regarding experimental setup in Sueca and Bisca, one team is composed of baseline agents while the rival team uses the experimental algorithm variation. Overall score and win rate is calculated for both teams. However, Hearts is not organized into teams, as each player tries to achieve victory by individual actions and score. Bearing this in mind, Hearts games were still initialized with two baseline agents and two other agents with an algorithm variation. The score and win rate of the baseline is an average of both baseline agents, as is the score and winning rate of the two agents with algorithm variations.

All charts are composed of bars with an error margin that was calculated by performing a One Sample T-Test with a 95% confidence interval on the mean of sampled results for each agent group with the same algorithm. Each figure is composed of 6 charts: the top 3 indicate the mean score for all three games, while the bottom 3 charts show mean win rates. In all experiments, results are shown as a grouped bar chart, where the baseline agent is displayed in a bright color, and plays against the experimental algorithm, displayed in a darker color. Note that directly comparing results between different algorithm variations (i.e. comparing different darker bars) does not allow to conclude that one variation is better than another, only that it performed better against the baseline.

6.1 Determinized UCT versus ISMCTS

The first experiments are focused on the two main algorithms for games with hidden information: Determinized UCT and ISMCTS. Results in figure 6.1 show that ISMCTS performs better at Sueca in all configurations. By increasing the number of determinizations, the performance decreases, which might indicate that Sueca is somewhat affected by strategy fusion. A possible explanation is that a generated card assignment greatly determines what is the current perception of the best move, but that move might in fact not be the overall best if we do not actually known the adversaries cards. Increasing the number of iterations per tree only improves win rate by a slight margin against ISMCTS.

In the case of Bisca, ISMCTS has significantly superior performance. Strategy fusion might not be the only factor that influences the result, since Determinized UCT produces low depth search trees. In a Bisca game, players focus on building a good hand until the deck ends, which happens in the end of the 11th move. Thus, 500 iterations per tree might not be enough to efficiently evaluate a move's success rate, since there is high branching factor 12 moves ahead. By raising the number of iterations, the winning rate is increased by around 3%, but still remains in the 28-33%. Strategy fusion can also be a likely negative factor in performance, since the player does not know beforehand what cards he can play (note that players take a new card after each trick). It might be the case that an agent develops a strategy accounting for the cards he is assuming to receive. For example, in a situation where the Ace of trumps is next in the deck, the agent will play a move assuming that the hidden Ace will be available for him to play next. However, this assumption is not realistic, as any card can be next in the deck.

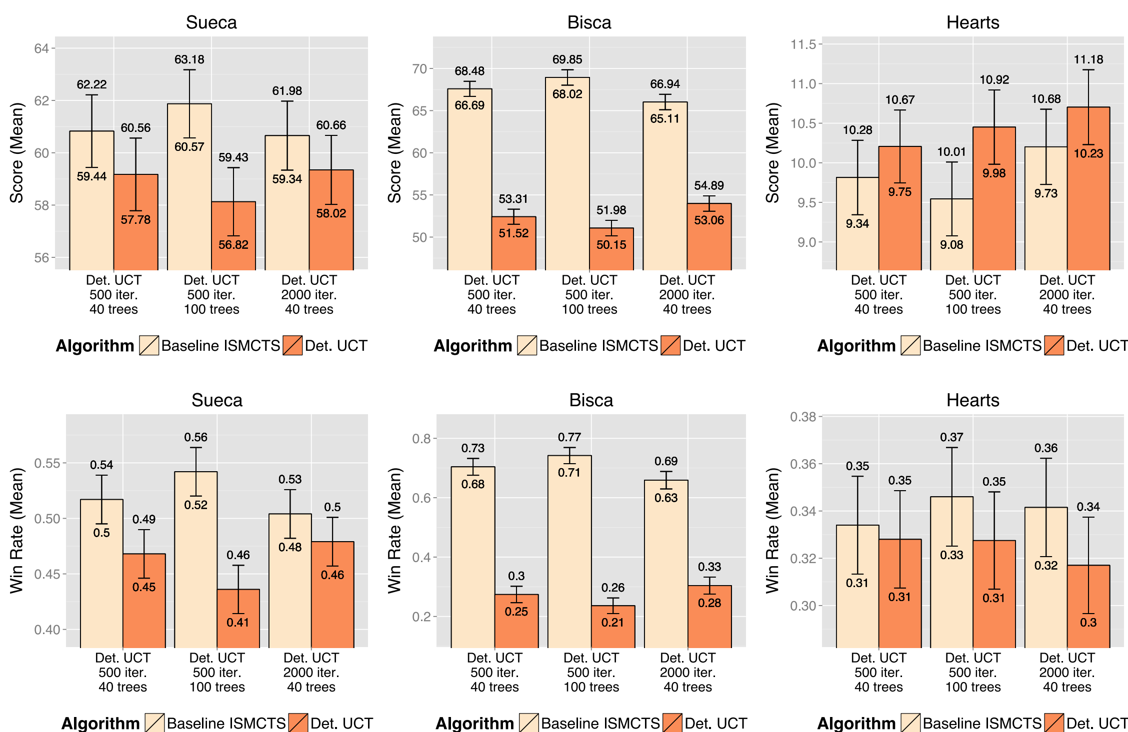


Figure 6.1: Results for Determinized UCT with different parameters for iterations and determinizations (trees) against a baseline ISMCTS with 10 000 iterations

For Hearts, ISMCTS does not provide a significantly better response against Determinized UCT. Tree depth or strategy fusion do not greatly affect the overall performance, and the score averages show that the baseline ISMCTS scores slighty better than Det.UCT, but only in a statistically significant manner when the number of determinizations in Det.UCT increases. This indicates that there might be an ideal configuration of iterations and determinizations to achieve a better play. However, note that simply scaling up these numbers does not necessarily increase efficiency, since 2000 iterations actually worsen the average score, possibly because a very high number of iterations causes the search to overfit the best move to a card that is specifically good against a determinized assignment of cards, but it is wrong to assume the adversaries will have that specific assignment.

6.2 Number of ISMCTS Iterations

In this section, the ideal number of iterations for ISMCTS is tested for each game domain against a baseline of 10 000 iterations. Increments of 10 000 iterations were tested up to 50 000. Note that performing 50 000 iterations takes at least 5x more time than searching with 10 000. But in reality, more search time does not necessarily translate into better play, as some games might not

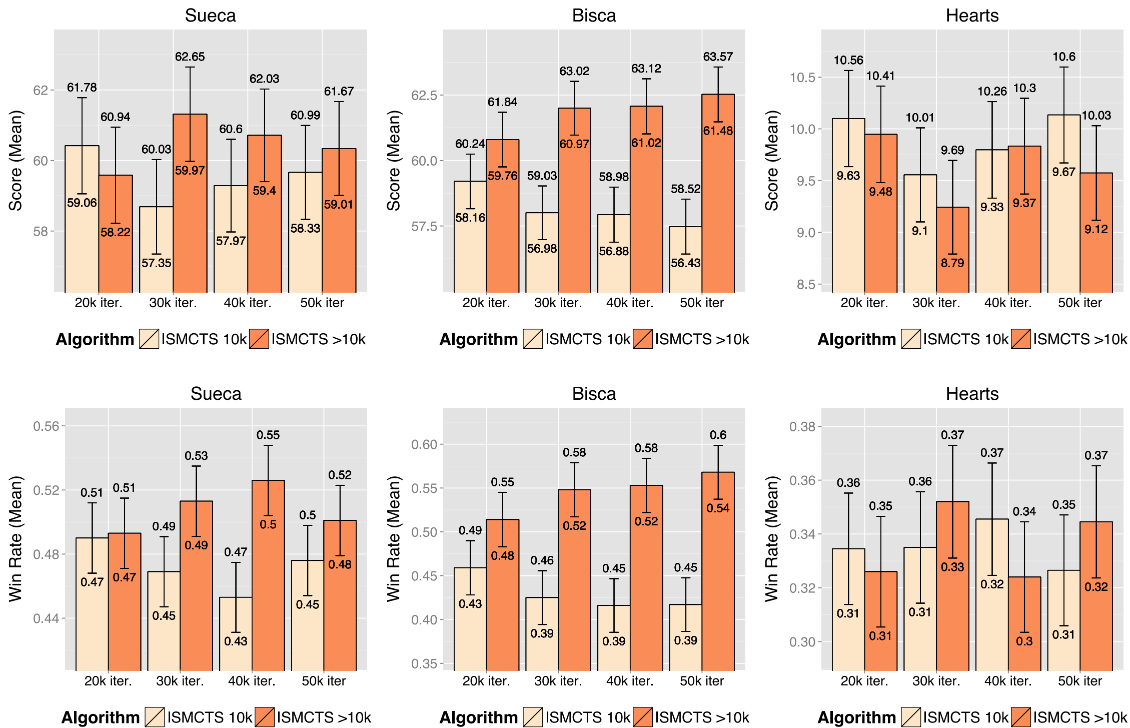


Figure 6.2: Results for an increasing number of iterations with ISMCTS against a baseline of 10 000 iterations.

need such a high number of iterations for convergence to optimal moves. Results are shown in figure 6.2.

For Sueca, using 20 000 iterations produces almost the same result as half the value, but 30 000 and 40 000 iterations make a very significant improvement in win rate. At the 50 000 level, performance seems to decrease.

In Bisca, increasing the number of iterations always improves the winning rate, leading to believe that a number above 50 000 iterations might yield even better results. This makes sense since good players are heavily focused on keeping good cards until the very end of the game, and agents with a higher move look ahead will definitely understand the benefits of this strategy.

In Hearts, scaling the number of iterations does not seem to make a steady difference. Best results where achieved with 30 000 iterations, but not with a statiscally significant margin for all tested number of iterations. This indicates that tree depth is not a winning factor in this specific game.

6.3 Random and Cheating Play

To better understand characteristics of the studied games, two factors are relevant: luck and knowing other players hands. These two factors will help establish upper and lower bounds: a random

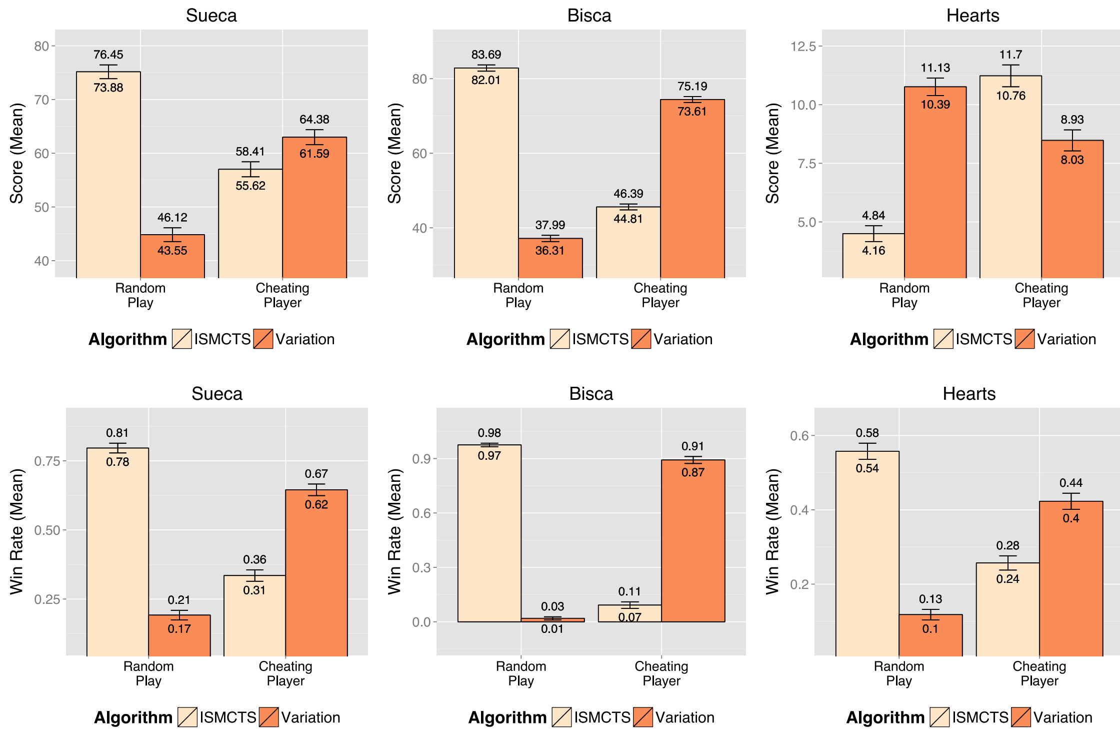


Figure 6.3: Results for a baseline ISMCTS of 10 000 iterations against a Random Play Agent and a Cheating Agent (with knowledge of every player card)

play agent will effectively demonstrate the worst possible average win rate (any lower would indicate the algorithm would be rewarding itself to lose). A player that can see adversary cards (effectively, a cheater) will limit the upper bound: knowing the definitive optimal move will still not guarantee victory, since luck greatly influences how many points one can score. Results are shown in figure 6.3.

In Sueca, a random play agent will guarantee a decent win rate: about 17-21%. Cheating will only allow for a 62-67% victory rate. This likely shows that there are a lot of games with very unbalanced card shuffling. For example, teams that start with very low ranked cards and trumps will never have a chance to win high valued tricks, even if the adversary only makes mistakes. There are some unofficial rules that allow players to cancel games if they do not have a certain amount of points in their starting hand that would reduce the influence of luck and allow to better evaluate player skill.

Bisca is a more strategic game, since random play only allows for a 1-3% win rate. On the other hand, cheating establishes a very high upper bound of 87-91% victory rate. This means that while luck somewhat influences the final score, players can still determine the outcome through skill and strategy.

Hearts is also very influenced by luck in card shuffling. Random play allows for 10-13% of victories and knowing adversary cards only enables about 40-44% chance to be the player with

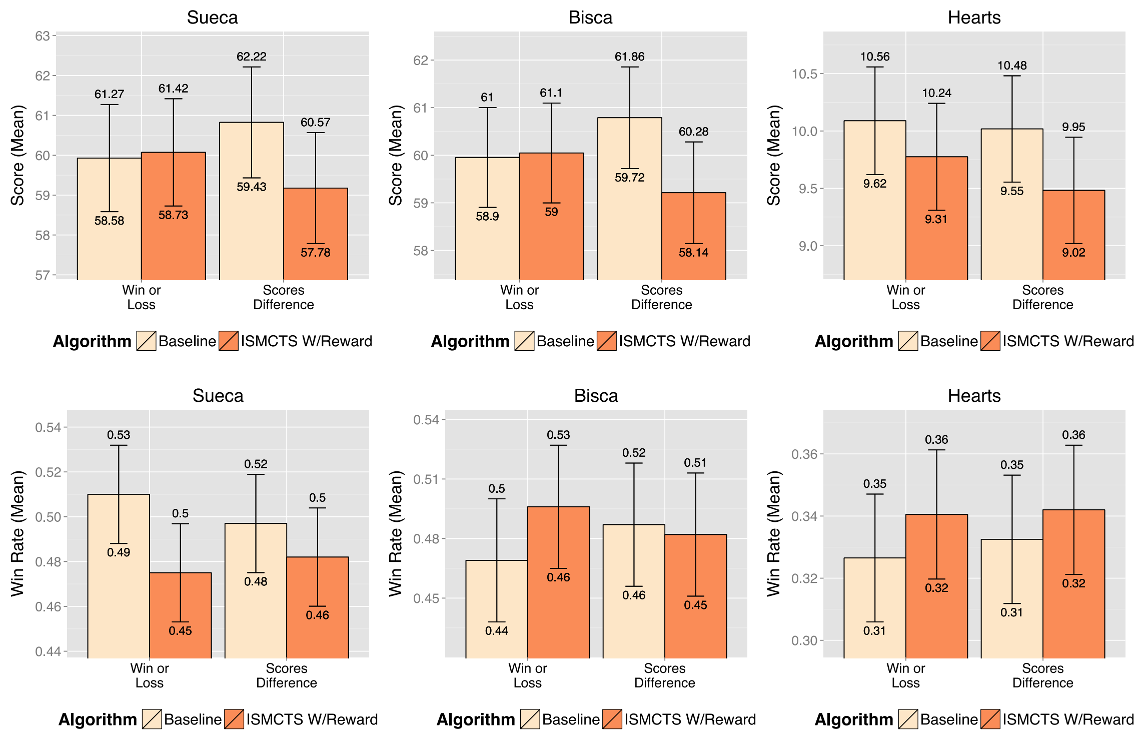


Figure 6.4: Results for reward function experiment with ISMCTS (10 000 iterations) using Positive Win or Loss (PWL) reward function as baseline. Alternatives against the baseline use Win or Loss (WL) function and Scores Difference (SD) function. Regarding the UCB1 exploration constant, the baseline PWL used 0.7 both Sueca and Hearts, and 0.25 for Bisca. WL used 1.75, 0.25 and 1.50 for Sueca, Bisca and Hearts, respectively, while SD used 0.50, 0.25, 0.25.

least score. However, the upper bound on the winning rate for an official Hearts game can actually be higher, since a game would significantly last more moves (i.e. until one player scores 100 points), minimizing the effect of luck.

6.4 Reward Functions

The reward functions used by the MCTS algorithms are a fundamental part of the tree search and greatly impact what is the current perception of the optimal move. Standard UCT Search usually defines rewards with the Positive Win or Loss function (PWL), where a loss would equal 0 and victories are valued as 1. Also, it has been demonstrated that the optimal UCB1 exploration constant is 0.7 for reward values between 0 and 1 [KSW06]. However, the definition of victory may not be so linear for certain games due to scoring systems.

Results for the initial analysis using the proposed Win or Loss (WL) and Scores Difference (SD) functions are visible in figure 6.4. Also, two experiments were conducted to find the best UCB1 constants in the range of $[0,2]$ for SD (fig. 6.5) and WL (fig. 6.6).

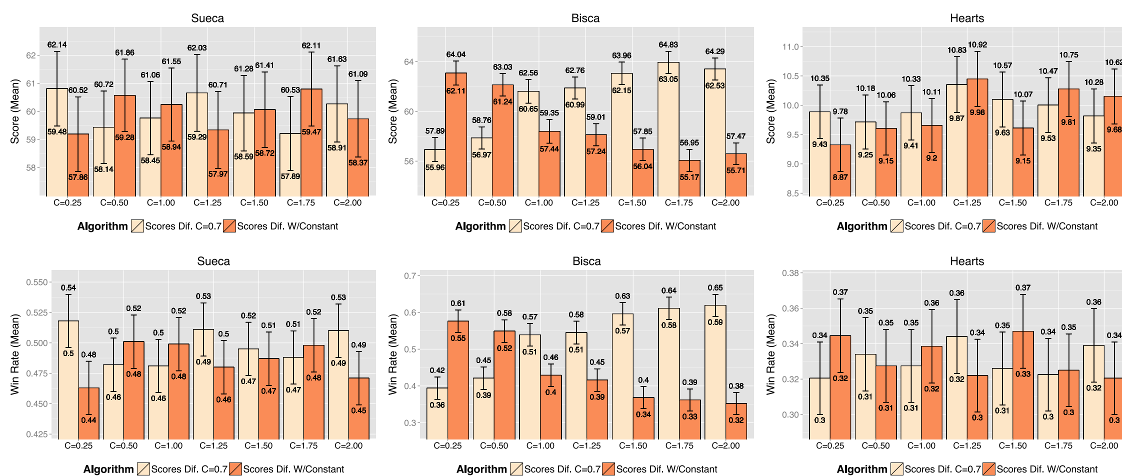


Figure 6.5: Results for best UCB1 exploration constant using Scores Difference reward function. The baseline agent uses 0.7 as the UCB1 exploration constant. All agents use ISMCTS with 2 500 iterations and Scores Difference as the reward function.

For Sueca, both WL and SD slightly worsen the win rate. A possible explanation is that good players should initially focus on winning and afterwards prioritize minimizing or maximizing the score difference (depending if they are the winner). With SD, maximizing score is the first priority, and as such, the agent will likely attempt greedy moves, and assumes the adversaries will also be greedy (i.e. risking high value cards to open chances of a victory by 2 or 4 match points). No result regarding the exploration constant gave significant differences, but 1.75 seems to be the best constant for WL, while SD reached good results with 0.50 and 1.00.

In the case of Bisca, WL gives a 3% advantage in win rate. The exploitation component of the UCB1 formula already takes into account the reward of a move with respect to the number of simulated games. With PWL, the exploitation value will equal to the simulated win rate of applying a given move. WL lowers the exploitation value and accentuates the difference between simulated wins and defeats, accelerating the UCT search convergence to moves that apparently have a superior win to loss ratio (while favouring exploration of unknown moves until a superior win to loss ratio is not found). In Bisca, this effect seems to be beneficial. With SD, win rates seem to be equal, however, the number of scored points is lower, possibly due to excessive greediness. Regarding constant values, the best found value was 0.25 for both functions. In fact, the use of a very low exploration value was even used for the baseline PWL, as using 0.7 significantly worsened the results. Exploitation of good moves seems to be the important in this game, possibly due to the fact that in the first 11 tricks, any card can be played. Exploring the use of most cards is fruitless, since only a fraction of the 9 cards in hand are relevant for the best move. Sueca and Hearts players must always follow the leading suit, and as such, exploration is restricted to a small set of relevant cards, effectively narrowing down the search. Bearing this aspect in mind, a constant value between 0 and 0.25 could possibly give better results.

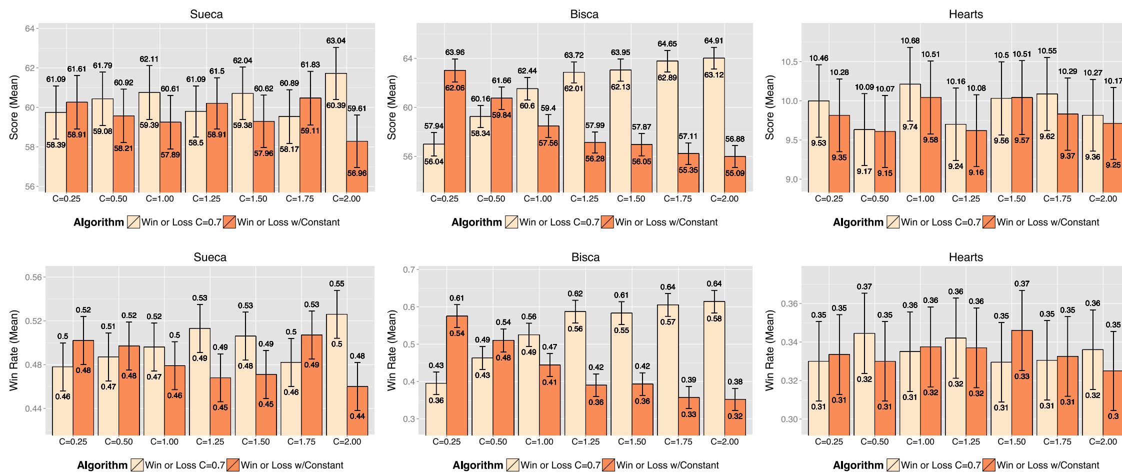


Figure 6.6: Results for best UCB1 exploration constant using Win or Loss reward function. The baseline agent uses 0.7 as the UCB1 exploration constant. All agents use ISMCTS with 2 500 iterations and Win or Loss as the reward function.

In Hearts, both WL and SD only improve win rate by 1%, but SD allowed to score less points, which is the main objective in an official game of Hearts. Regarding constants, no value gave significant differences, but 1.50 seems best for WL and 0.25 for SD.

A common problem is apparent in all tested games: random play takes over the end game decisions because victory is more often than not decided before the last move. But SD does not provide a good overall response to the win rate. We can hypothesize that the best reward function might be of hybrid nature. By using PWL or WL from the game start, the agent will attempt its best to win. After victory is decided, a SD function will try to maximize or minimize the difference, avoiding loss by a great score gap or a win by a small margin.

6.5 NAST Enhancement

The simulation phase is a very important step in the MCTS algorithm, and relies on a fundamental concept: that "the true value of an action may be approximated using random simulation" [BPW+12]. However, pure randomness can consider lines of play that will likely never happen if we assume the adversary always acts in a way that maximizes their utility, i.e. they play rationally. NAST changes the simulation phase in order to favour moves that have proven to be more sucessful, based on previous simulations. The calculation method to select the next simulated move is called a simulation policy, and in this section four different policies are tested: ε-greedy, Gibbs Distribution, Roulette Wheel and UCB1, with results visible in figure 6.7. Furthermore, NAST groups moves into sequences of varying length. In these experiments, n-grams of size 1, 2, 3 and 4 are tested, with results displayed in figure 6.8.

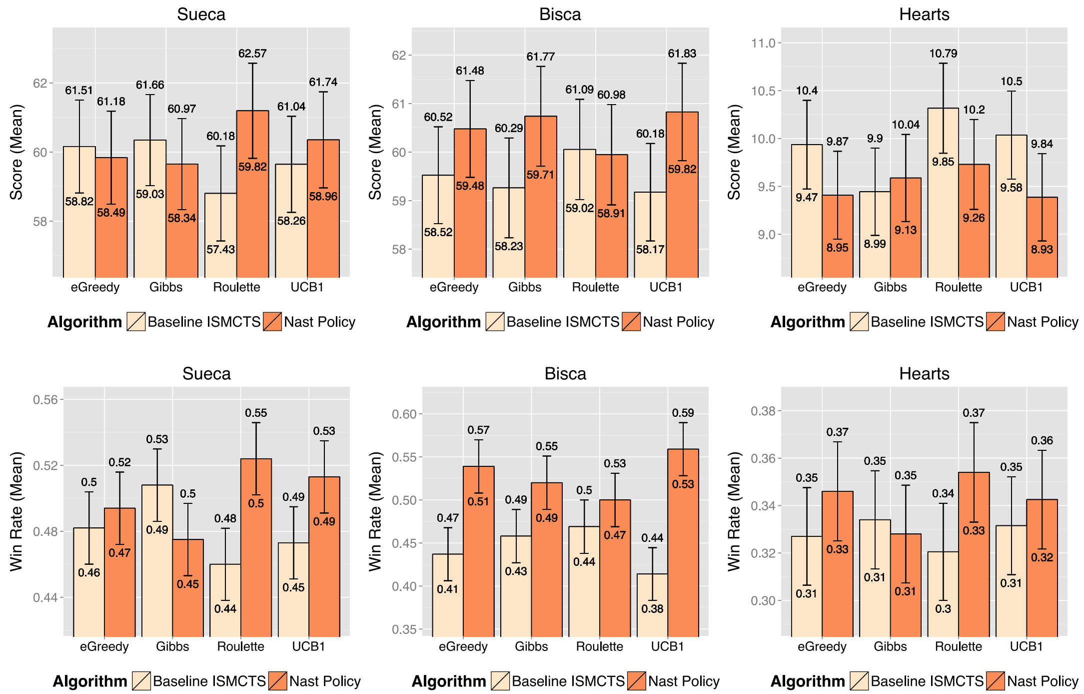


Figure 6.7: Results for best NAST simulation policy. All agents use ISMCTS with 10 000 iterations, with the baseline not using any enhancement. NAST agents use n-gram sequences of length 2 with different simulation policies, namely ε-greedy, Gibbs, Roulette, or UCB1.

Overall results show that NAST somewhat impacts performance in all three games. The UCB1 and Roulette Wheel policies yield better results than the baseline in Sueca, while UCB1, ε-greedy and Gibbs Distribution give a boost in performance for Bisca. In Hearts, Roulette seems to make the best improvement, but not by significant margins. Using n-grams of length 2 provides good results for Sueca and Bisca, while lengths of 2, 3 and 4 do not affect performance for Hearts in a statistically significant matter. One can argue that NAST would not improve playing performance because it is based on the assumption that good moves are somewhat independent of the context in which they are applied. Taking Sueca as an example, playing the Ace of trumps in the first round does not have the same effect as saving it for the last rounds (where it is likely applied best in most situations). However, note that simulations are purely random in the baseline ISMCTS, and this creates very unlikely and irrational situations. Imagine that a trick is lead with a high card, such as Ace of Spades (not the trump suit). It would make no sense for the next player to use a very high card, such as a 7, because there is a very low chance to actually win the trick this way. Using NAST of length 2, the sequence would be grouped as $\langle A\spadesuit, 7\spadesuit\rangle$ and a statistic would be stored, counting the number of times it appeared in a simulation, and how many simulations with this sequence turned out as a victory. The negative impact of the action would easily be perceptible by the simulation policy, since throwing away 10 points greatly hinders chances of success. With these statistics, simulations are biased toward the best found moves so far, but are still mixed with some randomness (depending on the chosen policy). Sueca and Bisca benefit most from the

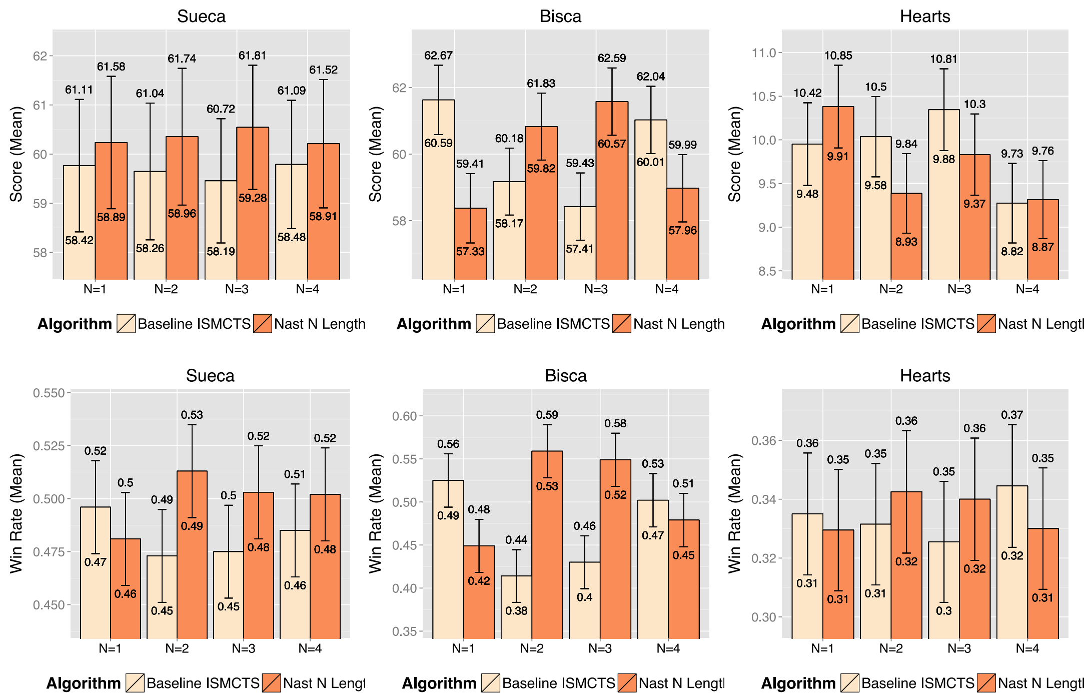


Figure 6.8: Results for best NAST N-gram length. All agents use ISMCTS with 10 000 iterations, with the baseline not using any enhancement. Nast agents use different number of N-grams length with the UCB1 simulation policy.

exploitation of good move sequences with exploration of untried sequences (UCB1), while Sueca also benefits from a random exploration of sequences, biasing the more sucessful ones (Roulette Wheel). While we only used the example of discarding high value cards, the reverse situation still gives good insight: $\langle 7\spadesuit, A\spadesuit\rangle$ would indicate that using an Ace is highly rewarding, and the agent would avoid a simulation that leads with a 7 since it has very low chances of winning. This eventually improves the quality of a simulation, since a reward for any given move would not have so much interference from such irrational moves.

Note that at least some historical information is still present: statistics for $\langle A\spadesuit, 7\spadesuit\rangle$ would only be considered if the Ace is held by the first player and the 7 is held by the second player. If one of these cards is already out of the game, then the statistics for this sequence would not even be taken into account. The available cards in the determinized game make up the eligible set of move sequences used to calculate the simulation policy.

While the overall result is positive, there might be some negative effects due to lack of contextual information. For example, examining the situation with $\langle 7\spadesuit, A\spadesuit\rangle$: we can not guarantee that the sequence is actually in the same trick. If the 7 was used as the last card in the previous trick, and the new trick is initiated with Ace, then there is a different interpretation of value. Many simulations would indicate that playing an Ace is a good response after a 7, but there is not much value in initiating a trick with an Ace after the 7 came out in the last trick. While this may not greatly influence the result, it could be mitigated by borrowing concepts of other simulation en-

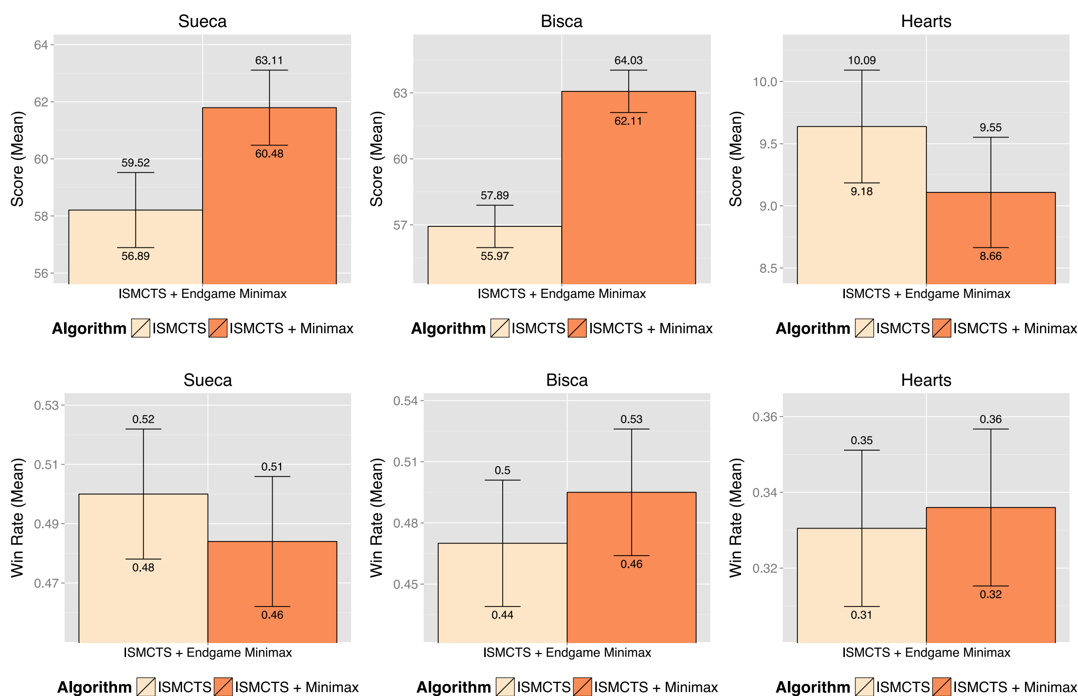


Figure 6.9: Results for hybrid use of Minimax and ISMCTS. All baseline agents use ISMCTS with 10 000 iterations, with the variant switching to Minimax search in the last 13 moves (Sueca and Hearts) or 15 moves (Bisca).

hancements such as EPIC, which groups sequences of moves into episodes that in this case are tricks.

6.6 Hybrid ISMCTS with Minimax

The main objective of this section is to test the effectiveness of applying Minimax in conjunction with ISMCTS. UCT Search is proven to eventually converge into the Minimax evaluated move, even if slowly [KSW06], but there is no guarantee for the case of non deterministic games. However, due to the computational complexity required by Minimax, we can only apply it near the end game, where only a reduced number of cards is left, and thus the best move can be evaluated in a feasible amount of time. Until that point is reached, ISMCTS is used instead. The upper bound time limit was established at around one minute. For Sueca and Hearts, responses under a minute in the experimental environment are reached with 13 moves left, while in Bisca this is achieved in 15 moves. Note that the heuristic function used with these games does not use any domain knowledge: a victory yields 1000 points, while defeat returns -1000 instead. The game score for the own agent is then added to this heuristic evaluation (higher scores are more valuable). As such, Minimax must run until the last move to properly calculate the score (i.e. with a depth level of 13 or 15). Also note that the game up to this point is still non deterministic for the case of Sueca and Hearts. With 13 moves left, there can be around 4200 different possible card assignments (10

cards left for 3 players, $\binom{10}{4} * \binom{6}{3} * \binom{3}{3} = 4200$), and a Minimax of depth 13 must run for each assignment. In the case of Bisca, the game is deterministic: after the deck runs out, one player can known exactly what the adversary holds, but branching factors are much higher, since each player has 7 or 8 cards when there are 15 moves left.

Results in figure 6.9 show that win rates do not change in statistically meaningful way. However, note that in all games the overall score improved significantly. This still can be explained by the same hypothesis detailed in section 6.4: the reward function used by ISMCTS is not the most appropriate. In the late game moves, victory is more often than not already decided, and players are simply trying to maximize the end score, but ISMCTS evaluates all moves as 100% reward or 0%, so any choosen move will not affect the game outcome. Also, note that many end game moves do not have any decision process involved: if the leading player uses a suit that others have, it is very likely that only 1 possible card can be played, and in that case, Minimax can not make a difference.

For the case of Bisca, note that win rate for Minimax increased slightly, even if not in a statistically significant manner. This gives some insight into a problem with the use of ISMCTS: after the last card of the deck is taken, the game becomes deterministic, since there is only one possible card assignment in the information set. This likely means that the hybridization technique would be best applied with a conjunction of ISMCTS with normal UCT Search once determinization is no longer required. From this experience, we can infer that Minimax in the final game phase does not make a meaningful impact, is significantly slower than ISMCTS, and a more appropriate reward function would likely minimize the score difference seen in the results.

Chapter 7

Conclusions

This final chapter presents the conclusions of this research. Initially defined goals and questions are answered according to the observed results, and an overview of guidelines for future research is given.

7.1 Goals

The initial goals defined in chapter 1.3 are now addressed as follows:

G1. Elaborate a study on the best known MCTS implementations applied in the context of trick taking card games.

All the research related to the topic is covered in chapter 2. Overall, it was concluded that much of the scientific experimentation with card games has surfaced in very recent years due to integration of determinization techniques with MCTS. In 2001, Ginsberg proposed the use of determinization to create an agent that challenged champion level human players in the card game Bridge [Gin01]. Later experiments showed that this approach was affected by shortcomings such as strategy fusion and non-locality, which compromised performance in certain card games, but still allowed champion play in others. A 2010 study on game characteristics allowed to better understand when determinization was best applied, supporting its use in trick taking card games [LSBF10]. ISMCTS was first proposed in 2012 [CPW12], as an alternative to Determinized UCT that removes effects of strategy fusion. Both ISMCTS and Determinized UCT have proven to be very effective algorithms in trick taking card games, with experimentation done in recent years for Hearts [PCW14], Dou Di Zhu, [PCW14] Spades [WCPR13b] and Skat [FB13]. Finally, in 2013, some enhancements such as NAST [PWC13] have been proposed and tested in card games, showing improved results for Hearts and Dou Di Zhu.

G2. Create a framework with some MCTS variations and enhancements, with application in card games.

The developed framework for card games is described in chapter 4.3, with a focus toward reproducible research. This design decision was carried out mainly due to the lack of required resources to reproduce results published throughout the researched scientific literature. While the framework might not satistify all developer needs, it is considered as a step in the right direction to aid other researchers in building on top of existing work and further improve the current state of the art artificial intelligence for card games and MCTS in general. Some ideas surfaced with the development of this framework, such as move replayability for faster debugging and hypothesis experimentation, as well as unit testing specific game states to understand and fine tune playing quality of agents before running long term experiments. As an end result, the framework can be used by other researchers to explore the datasets generated in the experimentation chapter, and allow them to run experiments of their own, with any modification to the source code.

G3. Develop a framework that enables different AI agents to compete against each other and evaluate relative performance and quality of play.

After researching possible projects, the Botwars framework (described in chapter 4.2) was choosen as the best alternative to acomplish the proposed goal. Some contributions were done to enable its use in this dissertation, namely the support of database storage for games and their full history of states. The framework serves as a solid foundation for researchers to run their experiments on, since it enables freedom of choice in the choosen programming language for agent development, allowing AI programs to compete against each other, independent of the language in which they are written. With the implementation of both playing and rendering logic, new games can be integrated into the framework, allowing for humans to play against developed AI's and easily coordinate long running experiments to evaluate the relative efficiency of each agent.

G4. Use the developed framework to experiment and analyse the best performing MCTS enhancements in a selection of three trick taking card games.

Both frameworks were used in conjunction to run the experiments described in chapter 6. Overall the proposed goal was achieved with success, since all experiments ran without any setbacks. Initially, the experiments were run in an iterative and manual way, but much of the setup process was later automated through bash scripts, while result aggregation and chart generation was automated with scripts using the R programming language. While some

observed results did not show statistically significant differences against the baseline algorithm, enough data was gathered to draw interesting conclusions for most of the cases.

Taking the achieved goals into account, the questions proposed in the beginning of this work are now answered as follows:

Q1. Which MCTS variations and enhancements are best applied to the trick taking card games Sueca, Bisca and Hearts?

Results obtained in experimental section 6.1 compare the use of two popular algorithms, Determinized UCT and ISMCTS. Although the three proposed games are somewhate similar in nature, intricacies in the gameplay of each game often lead to different results. For the case of Hearts, Determinized UCT gives very similar win rates, while ISMCTS shows better results in Sueca and Bisca. Another important aspect for both these algorithms is the reward function, which is evaluated in section 6.4. It is likely that all three games do not benefit from having a static reward function. While winning is the primary focus at the beginning, victories are almost always decided before the last move, which translates to poor playing quality when the outcome is already known. Win or Loss yields the best win rate in Bisca with very low exploration constants, while the standard Positive Win or Loss function gives the best win rates in Sueca and Hearts. The NAST simulation enhancement was also tested with different policies and move sequence lengths in section 6.5, with length of 2 proving to be the best alternative for all three games. Simulation policies have different effects for each game, as best results in Sueca, Bisca and Hearts were achieved with Roulette, UCB1 and Roulette, respectively.

Q2. Can an enhanced MCTS develop strong play against traditional AI techniques in trick taking card games?

As explained in section 2.5.2, traditional techniques such as Minimax are computationally intensive and can not be effectively applied without proper domain knowledge. It has been proved that MCTS converges to the Minimax solution [KSW06], but it is not possible to guarantee the same when applying determinization techniques. However, results visible in section 6.6, where Minimax is mixed with ISMCTS, show that the use of Minimax near the end game does not affect the outcome in a significant manner. Also, there is, to the best of our knowledge, no known scientific literature regarding rule based systems or game evaluation heuristics for Sueca and Bisca. As such, an enhanced ISMCTS with NAST of length 2 is so far the best researched aheuristic approach that can develop a good playing quality in these games.

Q3. Do the advantages of determinization outweigh its shortcomings when applied to trick taking card games? Can the shortcomings be efficiently diminished through specific enhancements?

As detailed in section 2.5.1.1, there are detrimental effects to search algorithms when applying determinization techniques, namely strategy fusion and non-locality. Games may have specific characteristics that minimize these negative effects, such as high leaf correlation and high disambiguation factor, i.e. the outcome does not easily change late in the game and hidden information is iteratively revealed as the game progresses (as explained in section 2.5.1.2). All three studied card games present these characteristics, but only experimental analysis can support a good use of determinization. Results in section 6.1 compare the use of a pure determinization approach (Det. UCT) against an information set approach (ISMCTS), which develops the full tree of game possibilities, effectively eliminating strategy fusion. Results show that pure determinization does not greatly affect agents playing Hearts, while Sueca and Bisca display more favourable results when using an information set approach, suggesting that the effect of strategy fusion may be considerable, specially in the case of Bisca, where difference in results is very significant.

7.2 Future Research

Some improvements can be done in the tested game domains. Specifically, the use of 3 different games required most enhancements to be independent of domain knowledge in order to make experimentation phase more feasible and reuse algorithm variations for all games. Even with overall positive results, enhancements tailored with domain knowledge for each game are of specific interest. Examples of such enhancements are the use of UCT Search with a state heuristic evaluation function, or using offline simulations to create a move database, which enables simulation policies to be initially more accurate than using online learning enhancements such as NAST. With use of the two developed frameworks, and the proposed determinization algorithm for trick taking card games, much of the required bootstrap work is reduced, allowing for future researchers to give a heavier focus toward one specific game and propose more custom tailored enhancements.

Also, some rules of Hearts were simplified in the context of this dissertation. Namely, the inclusion of card sharing in the beginning of each round and termination of the game once a player reaches 100 points. These simplifications led to a more regular number of moves and excluded the need to use enhancements that handle partially observable moves, which are not relevant for Sueca and Bisca. If more work is to be carried out specifically for the context of Hearts, then inclusion of the full rule set is an important aspect to bear in mind. Research on a more custom tailored score difference reward function for Hearts is also recommended, since adaptations were done to support testing a scores difference function that worked in all experimental game domains.

A negative aspect of Sueca is the reliance on luck, since highly unbalanced card distributions allow random playing strategies to achieve a high percentage of victories (about 17-21% in the conducted experiments). As such, the inclusion of an unofficial *"mandar abaixo"* rule is recommended, i.e. when one player has less than 10 points and no trumps, cards must be reshuffled and redistributed. This would discard very unbalaced games that do not reflect the quality of expert players.

References

[Abr91] Bruce Abramson. The Expected-Outcome Model of Two-Player Games. *The Expected-Outcome Model of Two-Player Games*, pages 29–73, 1991.

[ACbF02] P Auer, N Cesa-bianchi, and P Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[Bal83] Bruce W. Ballard. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence*, 21(3):327–350, 1983.

[BDS+04] Darse Billings, Aaron Davidson, Terence Schauenberg, Neil Burch, Michael Bowling, Robert Holte, Jonathan Schaeffer, and Duane Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Computers and Games*, pages 21–34. Springer-Verlag, 2004.

[BFT09] Ronald Bjarnason, Alan Fern, and Prasad Tadepalli. Lower Bounding Klondike Solitaire with Monte-Carlo Planning. *Proc. 19th Int. Conf. Automat. Plan. Sched.*, pages 26–33, 2009.

[BLFS09] Michael Buro, Jeffrey R. Long, Timothy Furtak, and Nathan R. Sturtevant. Improving State Evaluation, Inference, and Search in Trick-Based Card Games. *Proc. 21st Int. Joint Conf. Artif. Intell.*, pages 1407–1413, 2009.

[BPW+12] C B Browne, E Powley, D Whitehouse, S M Lucas, P I Cowling, P Rohlfshagen, S Tavener, D Perez, S Samothrakis, and S Colton. A Survey of Monte Carlo Tree Search Methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Bru93] B Brugmann. Monte Carlo Go. (ii):1–13, 1993.

[CBSS08] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. *Aiide*, pages 216–217, 2008.

[CJ07] Tristan Cazenave and Nicolas Jouandeau. On the Parallelization of UCT. *Monte Carlo Tree Search*, pages 93–101, 2007.

[CM07] Pierre-Arnaud Coquelin and Rémi Munos. Bandit Algorithms for Tree Search. *Arxiv preprint cs0703062*, 23(March):67–74, 2007.

[Cou06] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings Computers and Games 2006*. Springer-Verlag, 2006.

[CPW12] P.I. Cowling, E.J. Powley, and D. Whitehouse. Information Set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[CRW04] George Casella, Christian P Robert, and Martin T Wells. Generalized Accept – Reject sampling schemes. 45:342–347, 2004.

[CWH08] Guillaume M J B. Chaslot, Mark H M Winands, and H Jaap Herik. *Computers and Games: 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, chapter Parallel Monte-Carlo Tree Search, pages 60–71. Springer Berlin Heidelberg, 2008.

[dBT16] Casa do Benfica Teixoso. Regulamento do 5º torneio de sueca. Available at http://files.casadobenficateixoso.webnode.pt/200000051-05ece07d97/Regulamento_4MundialSueca_CasasBenfica.pdf, June 2016.

[Dra09] Peter Drake. The last-good-reply policy for Monte-Carlo go. *ICGA Journal*, 32(4):221–227, 2009.

[FB98] Ian Frank and David Basin. Search in games with incomplete information: a case study using Bridge card play. *Artif. Intell.*, 100(1-2):87–123, 1998.

[FB08] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)*, pages 259–264, 2008.

[FB10] Hilmar Finnsson and Yngvi Björnsson. Learning simulation control in general game-playing agents, 2010.

[FB13] Timothy Furtak and Michael Buro. Recursive Monte Carlo Search for Imperfect Information Games. *Proc. IEEE Conf. Comput. Intell. Games*, pages 225–232, 2013.

[FC09] Sergey Fomel and Jon F. Claerbout. Guest editors' introduction: Reproducible research. *Computing in Science and Engineering*, 11(1):5–7, 1 2009.

[Fer16] Pedro Fernandes. A simple monte carlo tree search library. https://github.com/pedrorfernandes/mcts, June 2016.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[Gin01] Matthew L. Ginsberg. GIB: Imperfect Information in a Computationally Challenging Game. *J. Artif. Intell. Res.*, 14:303–358, 2001.

[Gon16] Rui Gonçalves. Botwars, a framework and server for competitions of game bots. https://github.com/ruippeixotog/botwars, June 2016.

[GS07] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. *Machine Learning*, pages 273–280, 2007.

[GS11] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence*, 175(11):1856–1876, 2011.

[GW06] Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. *Twentieth Annual Conference on Neural Information Processing Systems NIPS 2006*, 2006.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, mar 1963.

[Hsu04] Feng-hsiung Hsu. *Behind deep blue : building the computer that defeated the world chess champion*. Princeton University Press, Princeton, N.J. Oxford, 2004.

[KSW06] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. Improved Monte-Carlo Search. *White paper*, (1):22, 2006.

[LSBF10] Jeffrey R. Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search. *Proc. Assoc. Adv. Artif. Intell.*, pages 134–140, 2010.

[Pag16a] Pagat. Briscola. Available at https://www.pagat.com/aceten/briscola.html, June 2016.

[Pag16b] Pagat. Classified index of card games. Available at http://www.pagat.com/class/, June 2016.

[Pag16c] Pagat. Hearts. Available at https://www.pagat.com/reverse/hearts.html, June 2016.

[Pag16d] Pagat. Sueca. Available at https://www.pagat.com/aceten/sueca.html, June 2016.

[Par90] David Parlett. *The Oxford guide to card games*. Oxford University Press, Oxford England New York, 1990.

[PCW14] Edward J. Powley, Peter I. Cowling, and Daniel Whitehouse. Information capture and reuse strategies in Monte Carlo Tree Search, with applications to games of hidden information. *Artificial Intelligence*, 217:92–116, 2014.

[PWC11] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. Determinization in Monte-Carlo Tree Search for the card game Dou Di Zhu. *Proc. Artif. Intell. Simul. Behav.*, pages 17–24, 2011.

[PWC13] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search. *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2013.

[RN10] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[RSS11] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. On the Behavior of UCT in Synthetic Search Spaces. *Proc. 21st Int. Conf. Automat. Plan. Sched.*, 2011.

[SCPW12] N. Sephton, P.I. Cowling, E.J. Powley, and D. Whitehouse. Parallelization of Information Set Monte Carlo Tree Search. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):120–143, 2012.

[Stu08] Nathan Sturtevant. An analysis of UCT in multi-player games. *ICGA Journal*, 31(4):195–208, 2008.

[SWU12] Jan A. Stankiewicz, Mark H. M. Winands, and Jos W. H. M. Uiterwijk. *Monte-Carlo Tree Search Enhancements for Havannah*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[TWB12] M. J. W. Tak, M. H. M. Winands, and Y. Bjornsson. N-grams and the last-good-reply policy applied in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):73–83, June 2012.

[WCPR13a] Daniel Whitehouse, Peter I. Cowling, Edward J. Powley, and Jeff Rollason. Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. *Proc. Artif. Intell. Interact. Digital Entert. Conf.*, pages 100–106, 2013.

[WCPR13b] Daniel Whitehouse, Peter I. Cowling, Edward J. Powley, and Jeff Rollason. Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game. *Proc. Artif. Intell. Interact. Digital Entert. Conf.*, pages 100–106, 2013.

[Whi14] Daniel Whitehouse. *Monte Carlo Tree Search for games with Hidden Information and Uncertainty*. PhD thesis, University of York, 2014.

[WPC11] Daniel Whitehouse, Edward J Powley, and Peter I Cowling. Determinization and information set Monte Carlo Tree Search for the card game Dou Di Zhu. In *2011 IEEE Conference on Computational Intelligence and Games, CIG 2011*, pages 87–94, 2011.