Monte Carlo Tree Search for games with Hidden Information and Uncertainty

Daniel Whitehouse
PhD
University of York
Computer Science

August, 2014

Abstract

Monte Carlo Tree Search (MCTS) is an AI technique that has been successfully applied to many deterministic games of perfect information, leading to large advances in a number of domains, such as Go and General Game Playing. Imperfect information games are less well studied in the field of AI despite being popular and of significant commercial interest, for example in the case of computer and mobile adaptations of turn based board and card games. This is largely because hidden information and uncertainty leads to a large increase in complexity compared to perfect information games.

In this thesis MCTS is extended to games with hidden information and uncertainty through the introduction of the *Information Set MCTS* (ISMCTS) family of algorithms. It is demonstrated that ISMCTS can handle hidden information and uncertainty in a variety of complex board and card games. This is achieved whilst preserving the general applicability of MCTS and using computational budgets appropriate for use in a commercial game. The ISMCTS algorithm is shown to outperform the existing approach of *Perfect Information Monte Carlo* (PIMC) search. Additionally it is shown that ISMCTS can be used to solve two known issues with PIMC search, namely *strategy fusion* and *non-locality*. ISMCTS has been integrated into a commercial game, Spades by AI Factory, with over 2.5 million downloads.

The *Information Capture And ReUSe* (ICARUS) framework is also introduced in this thesis. The ICARUS framework generalises MCTS enhancements in terms of information capture (from MCTS simulations) and reuse (to improve MCTS tree and simulation policies). The ICARUS framework is used to express existing enhancements, to provide a tool to design new ones, and to rigorously define how MCTS enhancements can be combined. The ICARUS framework is tested across a wide variety of games.

Contents

List of Figures

List of Tables

List of Algorithms

Preface

In 2010 I graduated from the University of Manchester with a Masters degree in Mathematics. Whilst an undergraduate I shared a house with several computer science undergraduates and was often involved in discussions about the subjects they were learning. I have always been interested interested in computers, ever since having used an Amstrad CPC 464 when I was young (which had 64 kb of memory, a command line interface, tape storage and supported programming in BASIC). Whilst studying mathematics I found myself drawn towards topics such as numerical analysis, computational complexity and graph theory. Additionally I was taught `C++` for scientific computing and implemented Conways Game of Life. I am a big fan of computer games particularly real time strategy games such as the Command and Conquer series, Total Annihilation and Supreme Commander. I also grew up regularly playing board and game card with my family and have casually played several collectible card games.

When I saw advertised an opportunity to do a PhD developing Artificial Intelligence for the card game Magic: The Gathering, I found it immediately appealing both as a way to move from mathematics into the area of computer science and as a way to indulge in my passion for games. Luckily I was awarded the position and this thesis is one of the main outcomes. When I started this project I had much to learn, having had no previous training in computer science or artificial intelligence and being unfamiliar with Monte Carlo Tree Search, which became the focus of my work. I was able to learn very quickly and my background in mathematics was very useful in picking up many of the theoretical ideas. Additionally I had the opportunity to attend many conferences, which exposed me to a wide variety of the areas of work done in the field of artificial intelligence and the opportunity to engage in conversations with experts from different fields.

I have enjoyed doing this work and have learnt many new skills in a short period of time. I am also pleased with the outcome, having published 9 papers (with more under review), worked with the company AI Factory to integrate my research into their products and created AI for the physical travelling salesman competition (which won three international competitions by a large margin). All of these accomplishments would not have been possible without the help, support and guidance of many people whom I would like to thank.

I would like to thank my supervisor Peter Cowling. Firstly for providing me with the opportunity to do a PhD in this area and also for training me as a researcher, scientist, and author. He has offered frequent and valuable advice and without his input this thesis would not exist. I would also like to thank my co-supervisor Edward Powley, with whom I discussed my work on a daily basis and frequently collaborated with. His help and instruction has elevated me from the level of a beginner programmer to being confident and proficient with many different programming languages and technologies. I would also like to thank both of my supervisors for their help with with proofreading my thesis.

I would like to thank Jeff Rollason, CEO and Founder of AI Factory. He provided code for the AI in several of their products which was a useful testbed.

Additionally he made many useful suggestions and insights which guided the research presented in this thesis. I would also like to thank Néstor Romeral for many useful insights and discussions, as well as being an excellent host for our project meeting in Zaragoza, Spain.

I would like to thank several people whom I have collaborated with whilst competing this thesis. Firstly I would like to thank Cameron Browne for the massive task of organising the MCTS survey paper [1]. I would also like to thank Diego Pérez, Simon Lucas, Spyridon Samothrakis and Philipp Rohlfshagen for their contributions to organising the physical travelling salesman competitions, and for providing us with the opportunity to collaborate with them publishing the results as well as everybody else involved in the "UCT for games and Beyond" project. I would like to thank Sam Devlin for his contributions to analysing the data given to us by AI Factory. I would also like to thank the many people I have shared offices with or played games with at lunchtimes including Colin Ward, Paul Trundle, Stephen Remde, Nick Sephton, Philip Mourdjis and Yujie Chen for advice and interesting (or at least entertaining) discussions.

I would also like to thank my friends Anna Totterdill, James Short and Andy Wears for their support as well as my bandmates for providing me with an excellent creative outlet (and putting up with my pedantry) and all of these people simply for being great to be around. Finally I thank my family for their support, including my parents Michael and Gill, my sister Jennifer as well as Tom and Chris Messruther, and John and Dawn Stait.

Acknowledgements

This research has been carried out in the facilities of the University of Bradford and the University of York. This research was funded as part of the EPSRC project "UCT for Games and Beyond" (EPSRC Reference: EP/H048707/1 and EP/H048707/2).

Author's Declaration

The work presented in this thesis is drawn from several papers, both published and under review. Table 1.1 (page 11) lists all published work which is presented in this thesis along with which chapter the work appears in and a summary of my contributions to the work. Table 1.2 (page 11) list all papers which are submitted and under review from which work in this thesis is drawn, along with which chapter the work appears in and a summary of my contributions to the work. The work presented in this thesis was carried out as part of a joint research project, where I contributed original ideas, actively participated in all discussions and meetings, and co-authored both the research code and written publications. A summary of my individual contributions to the project is presented in Section 1.2 (page 6). This work has been submitted exclusively for PhD examination at the University of York.

Chapter 1

Introduction

The topic of this thesis is the application of *Monte Carlo Tree Search* (MCTS) algorithms to games with hidden information and uncertainty. MCTS is a family of algorithms in the field of artificial intelligence which first emerged in 2006 [2, 3, 4] most notably the UCT algorithm [2]. MCTS has contributed to advances in the fields of computer Go [5, 6, 7], General Game Playing [8, 9] and many other domains as well as attracting a substantial amount of interest in the research community summarized in a survey by Browne et al [1]. The majority of research to date on MCTS has focused on two player zero sum games with perfect information. However many popular games (and real world problems which can be modelled as games) have elements of imperfect information which results in complex (large or infinite) implicitly defined state-action graphs. This thesis focuses on the issue of information asymmetry between players, as might arise in a card game for example.

Developing AI for games with imperfect information provides an important testbed for the investigation of tree search algorithms in AI, as well as addressing the needs of games developers particularly in the emerging market of adapting traditional table top games and eurogames to mobile platforms. In order to target these devices bespoke knowledge based systems are developed for each game and often these agents ignore hidden information (i.e. cheat), which can lead to unsatisfying player experiences. This thesis demonstrates how MCTS can be adapted to games with imperfect information in order to take advantage of the same useful properties of MCTS which have led to advances in complex domains with perfect information. This leads to *Information Set Monte Carlo Tree Search* (ISMCTS) family of algorithms, which can be used to create engaging play in complex games with hidden information and uncertainty using a computational budget appropriate for a commercial game (for example running on a mobile device) and without utilizing domain knowledge (other than the game rules).

Two main original contributions are presented in this thesis. Firstly the *information set MCTS* (ISMCTS) family of algorithms is introduced, which adapt the MCTS algorithm to search a tree of information sets rather than a

tree of states through the use of determinizations (randomized assignments of hidden information). In an information set tree each node represents a set of states a player believes the game may be in, rather than an explicit state. The ISMCTS algorithm contributes a variety of different ways to share statistics between information sets, by using nodes which represent many information sets. This allows good decision making in large complex games where using a unique node for each unique information set is intractable. ISMCTS represents a significant advance over existing methods for games with imperfect information, in particular ISMCTS addresses the issues of strategy fusion and non-locality. Strategy fusion arises when the best decision to make may be different depending on which state a game is in and the strategy for each state must be somehow "fused" into a single strategy to each possible state. Non-locality arises when the past decisions of players are not taken into account when determining which states are likely (or even possible). ISMCTS is shown to produce a strong level of play in several games without using any domain specific knowledge, preserving the general game playing capability of MCTS for perfect information games whilst using a small computational budget. Finally it is demonstrated that ISMCTS can be adapted to exhibit bluffing and inference, which is not possible in existing determinization based approaches.

Secondly the *Information Set Capture and Re-Use* (ICARUS) framework is introduced, which facilitates the definition of enhancements to both MCTS and ISMCTS. The ICARUS framework unifies the many different variations of MCTS algorithms by defining MCTS as an algorithm which performs a series of simulations from which information is captured and then subsequently re-used to improve future simulation in a process equivalent to reinforcement learning. All existing MCTS enhancements can be expressed in the ICARUS framework and it can be used to invent new ones. as well as to express new ones. Furthermore the ICARUS framework defines combination operators on enhancements, which provides the first mechanism for precisely specifying how different enhancements are combined (without having to study implementation). One significant application of this is a new and elegant MCTS algorithm which uses bandit policy selection for all decisions in simulated games.

The work in presented in this thesis was carried out as part of an EPSRC funded project "UCT for Games and Beyond"[1]. To date the work on this project has resulted in 9 publications in peer reviewed journals and conference proceedings (listed in Table 1.1). with a further 5 submitted and under review (listed in Table 1.2). Furthermore the ISMCTS algorithm has been integrated into a commercial game, Spades by AI Factory which has had over 2.5 million installs [10]. The ISMCTS algorithm offered a significant improvement over the existing heuristic based AI in the game, particularly in the challenging case of nil-bids. Recent ongoing work has focused on enhancing the ISMCTS algorithm in order to produce more intuitive behaviour and improve the player experience. Finally as part of the project MCTS was applied to a real-time problem, the Physical Travelling Salesman (PTSP) competition. MCTS proved

[1]EPSRC Reference: EP/H048707/1 and EP/H048707/2

to be the leading approach winning the competitions in 2012 and 2013 as well as the multiple objective PTSP competition in 2013. This work is not directly relevant to the main subject of this thesis so is not described here.

This chapter is structured as follows. Firstly Section 1.1 introduces imperfect information in games, AI in games, and areas the work presented in this thesis advances. Section 1.1.1 introduces MCTS algorithms along with the research and applications of MCTS. Section 1.2 presents an overview of the EPSRC funded project "UCT for Games and Beyond". The work presented in this thesis formed part of the hidden information and uncertainty strand of this project. Section 1.3 lists the outcomes of this project including the original contributions presented in this thesis and other related work. Finally Section 1.4 presents an overview the structure of this thesis and the content of each chapter.

1.1 Background

The problem of designing artificial intelligence (AI) agents for deterministic games with perfect information is one of the oldest problems in Computer Science, with seminal works by von Neumann [11], Turing [12] and Shannon [13] among others. The most successful approach to artificial intelligence for games is game tree search. Other techniques such as reinforcement learning and neural networks have been used often in combination with tree search [14]. In a game tree nodes correspond to states the game can be in, and edges correspond to actions made by players. The edges have direction and so the game tree is strictly a directed acyclic graph where the nodes are in one-to-one correspondence with the states of the game (in the game tree the same state may correspond to multiple nodes reached by different sequences of actions). The initial (or current) state of the game is the root node and leaves correspond to states in which terminal conditions are met (i.e. the game is over). A game tree is then a map of possible future states and allows an agent to reason about the consequences of decisions.

Much of the work on artificial intelligence has focused on combinatorial games where no information is hidden and all actions are deterministic. This includes games such as Go, Chess and Tic Tac Toe, as well as many others (including single player puzzles). However many modern games feature elements of hidden information and uncertainty (non-determinism). Hidden Information occurs when a player cannot observe some information about the state of a game, more precisely a player knows the game is in one of a set of many possible states called an *Information Set*. In a card game for example a player may know an opponent holds 7 cards in hand, but not which 7 cards, instead observing an information set whose states correspond to every possible set of 7 cards the opponent could legally hold in hand. Uncertainty occurs when a player makes a decision which has a random outcome, or more precisely a player makes a decision and the game transitions into one of several possible states according to a probability distribution over those states. For example if a player takes the action of rolling a six sided die, the state of the game transitions into

one of six possible rolls each with equal probability.

Hidden Information and Uncertainty introduce new decision making concerns which do not exist in combinatorial games. Firstly when games are non-deterministic (as in an action can lead to several potential outcomes), players must apply probabilistic reasoning to maximize the probability they will win a game (or maximize their expected reward). However the optimal decision may not always produce the best possible reward. Secondly when information is hidden, players may deduce that some states are more likely than others by observing opponents' actions in a process known as inference. Correspondingly, if players are performing inference then other players can counter by bluffing in order to trick other players into making incorrect deductions about which states are likely. Consequently the opponent's strategies may also be a source of uncertainty, since they may vary decisions to avoid being predictable (known as using a mixed strategy). Hidden Information and Uncertainty are used in games to better model real-life decision-making and can also be used to create a balance of luck and skill. For example in a combinatorial game such as chess an expert player might always beat a novice, which is not entertaining for either player but in a non-deterministic game such as poker, the novice player might have a small chance to win, resulting in closer and more interesting game.

AI for imperfect information games is less studied than for combinatorial games. This is in part because imperfect information adds a large amount of complexity and because there are combinatorial games where humans are still better than the best computer agents, for example Arimaa [15] and Go [7]. However many AI techniques for imperfect information games exist, some of which are based on tree search for perfect information games. For example a large class of algorithms (including the new methods described in this thesis) make use of *determinizations*. A determinization is a state from an information set assumed to be the state of the game. Existing AI techniques for combinatorial games can then be applied to a determinization in order to make a decision. Determinization based approaches have been hugely successful in games such as Bridge [16] and Scrabble [17].

The goal of AI research has often been to produce optimal or human beating levels of performance in games. However in commercial games, the goals for AI are substantially different. Firstly commercial games aim to entertain players and thus AI agents must be fun to play with and cater to a range of player abilities (not just performance at the highest level of skill). Additionally AI agents are often designed to behave in a manner that is intuitive for human players to understand. Finally AI in commercial games must be created within a budget for implementation and testing time and utilize modest amounts of processor and memory resources. Some games utilize techniques for combinatorial games such minimax search. In imperfect information games it is not uncommon for AI agents to "cheat" and know hidden information, which can lead to an unsatisfying experience for players. To this end commercial AI for imperfect information games is sometimes based upon bespoke rule based systems for a particular game which can be made to run quickly and consume small amounts of memory. This enables the behaviour of the AI to be precisely controlled by

a designer. The drawbacks of this approach are large development and testing requirements, lack of re-usability between games, and strange behaviours in situations unforeseen by the rule designer. Furthermore the AI must be modified and retested even when small changes are made to the game rules.

1.1.1 Monte Carlo Tree Search

Using Monte Carlo simulations to evaluate states was first proposed by Abramson in 1990 [18], but was not widely used (probably due to the large successes of the minimax algorithm in widely studied domains such as Chess [19] and Checkers [20]). *Monte Carlo Tree Search* (MCTS) is a family of tree search algorithms which was not described until 2006 [4, 2] and greatly improves the use of Monte Carlo simulations by applying the concepts of exploitation and exploration to build a search tree which guides simulations. MCTS is most notable for leading to large advances in the field of computer Go [5, 6, 7]. Furthermore MCTS is the leading technique in many domains for example Hex [21] and General Game Playing [8, 9]. MCTS has been applied to a wide variety of domains and problems and has attracted a large amount of interest from both researchers [1] and game developers [10]. One of the contributions of the work done in this project was a comprehensive survey of MCTS research up to 2011 [1] which covered different variations of and enhancements to MCTS as well as a comprehensive list of problems to which MCTS has been applied. More recently MCTS has been successfully applied to real-time games including successes in Ms Pac Man [22, 23] and PTSP competitions [24, 25]. There has also been work on the application of MCTS to games with hidden information and uncertainty, including the work presented in this thesis. A survey of MCTS research is presented in Chapter 3.

One of the most significant benefits of MCTS is the algorithm does not require domain-specific knowledge beyond the rules of the game which specify which actions are possible from each state and which states are terminal. This makes MCTS readily applicable to any domain that may be modelled using a decision tree (although adding domain-specific knowledge can be beneficial). Therefore MCTS can be described as an *aheuristic* algorithm. MCTS is also an iterative algorithm which means that all values are always up-to-date following every iteration of the algorithm. In practice this means the MCTS algorithm is able to return an action at effectively any moment in time since each iteration is usually short compared to the overall run time of the algorithm. MCTS can be run for additional iterations which often improves the result. MCTS can therefore be described as an *anytime* algorithm. MCTS also builds a partial subtree of a complete game tree, and the addition of nodes is biased towards more promising states. This leads to an *asymmetric* sub-tree of the game tree being built up over over time, in contrast to minimax search which expands an entire subtree to a fixed depth. In other words, the building of the partial tree is skewed towards more promising and thus more important regions and less time is spent exploring less promising regions.

1.2 UCT for Games and Beyond

The work in this thesis was carried out as part of a 3-year EPSRC funded project entitled "UCT for games and Beyond" (EPSRC Reference: EP/H048707/1). This project commenced in 2010 and consisted of three strands:

- MCTS for games with Hidden Information and Uncertainty at the University of York
- MCTS for real-time games at the University of Essex
- MCTS for computational creativity at Goldsmiths, University of London

The work presented in this thesis was carried out as part of the MCTS for games with Hidden Information and Uncertainty strand which was carried out by myself, Dr. Edward Powley and Prof. Peter Cowling. Initially I was to work mostly independently under supervision, but the project gravitated towards more collaborative methods of working. The design of algorithms and experiments as well the interpretation of results was discussed at weekly meetings, then smaller tasks were assigned to each individual. Myself and Dr. Edward Powley shared responsibility for implementation and testing of all code. I actively participated in discussions on all aspects of the project as well as introducing a large number of the ideas. Additionally I co-authored every publication, and shared approximately equal authorship responsibilities (regardless of the orders in which authors are listed).

The aim of this project was to answer the question: **How can MCTS deal with uncertainty and hidden information?** For the majority of problems where decision tree models are appropriate, particularly real-time video games, the tree is nondeterministic, with both hidden information and randomness. In almost all of the games where MCTS has been used successfully so far, the decision tree is deterministic, and all players have complete information about the game state. The presence of chance nodes in the tree due to random outcomes or hidden information adversely affect standard implementations of MCTS, since the Monte Carlo selection mechanism will prefer lucky outcomes to unlucky ones if it used to select chance outcomes. The decision which is then propagated to the current position at the top of the tree is the one which should be taken given best possible luck. However, for most games (and other situations modelled by decision trees) this decision is hopelessly overoptimistic. Consider playing Monopoly with the assumption that you never land on one of your opponents properties, and they always land on yours. A more reasonable approach (which is used in practice) would be to insist that all outcomes from a chance node are sampled according to some sensible prior distribution. However, this approach breaks down due to the massive increase in branching factor. Consider playing a 2-player card game where the opponent holds 2 unseen cards from a standard deck. A naïve approach has a chance node with ${}^{52}C_2 = 1326$ outcomes (one for each pair of cards the opponent could be holding), so that even for this simple situation the problem is 3 orders of magnitude harder than the deterministic case.

In this thesis MCTS is shown to be a promising approach for games with imperfect information, by exploiting the useful properties of MCTS to handle different sources of uncertainty and hidden information. Imperfect information leads to combinatorial explosions in the size of game trees, which is an issue for fixed depth approaches since only a shallow search can be performed on even a large computation budget. The asymmetric nature of MCTS provides a mechanism for overcoming this limitation and additionally MCTS has shown to be effective in large branching factor games such as Go. Given the large combinatorial nature of imperfect information games, finding exact solutions is intractable, however MCTS is shown to produce plausible playing strength given practical amounts of computation resources in many different games. Finally this project aimed to preserve as much as possible the aheuristic nature of MCTS in applications to imperfect information games. In each of the games studied in this thesis either little or no knowledge was used.

1.3 Research Hypothesis and Project Outcomes

In this section the outcomes of the imperfect information games strand of the UCT for games and beyond project is summarized, with the goal of answering the question how can MCTS deal with uncertainty and incomplete information? The research presented in this thesis addresses the following four hypotheses:

Hypothesis 1: Uncertain outcomes and unknown states can be represented using an MCTS tree In this thesis, the *information set MCTS* (ISMCTS) family of algorithms [26, 27] is introduced which adapt the MCTS to search trees of information sets rather than trees of states. ISMCTS handles hidden information through the use of determinizations of information sets, where a determinization is a sampled state from an information set assumed to be the actual state of the game. ISMCTS improves upon earlier determinization based approaches such as perfect information Monte-Carlo search [28, 29, 30]. ISMCTS is extended to handle partial observability with the MOISMCTS algorithm [27] (which is equivalent to ISMCTS when there is no partial observability).

Hypothesis 2: The issues of strategy fusion and non-locality with determinization based approaches can solved using MCTS Since the ISMCTS algorithm is determinization based, issues with determinization such as strategy fusion and non-locality must be addressed. Strategy fusion is an issue with determinization based algorithms where it is incorrectly assumed a player can alter their decisions with respect to information they cannot know. Non-locality occurs when a state must be impossible if it is assumed players are behaving rationally (for example taking moves which lead to an immediate win), but is considered to be a possible determinization of an information set. In Chapter 5 experiments estimated the impact of these problems for the card game Dou Di Zhu and showed that strategy fusion in particular is a large problem. Strategy fusion is handled by ISMCTS through the use of information set trees.

The problem of non-locality is addressed through the use of inference and opponent modelling in Chapter 7. Whether non-locality is an important problem varies from game to game, since experiments for Dou Di Zhu showed that inference is not hugely beneficial, whereas inference is a critical part of strategy in the game The Resistance for example.

Hypothesis 3: Inference and bluffing can be modeled using MCTS It has been demonstrated that inference is possible with ISMCTS through biasing the distribution of determinizations sampled (See Chapter 7). However this requires a good opponent model in order to be effective. In Chapter 7 it is explained how ISMCTS can be used as an opponent model for inference in place of an external model, preserving the aheuristic nature of ISMCTS. Furthermore it is demonstrated that ISMCTS can learn bluffing behaviour when determinizations of opponents information sets are searched. Experiments presented in Chapter 7 measured the effect of "perfect" inference in the card game Dou Di Zhu and it was concluded that inference was not a significant barrier to good playing strength in that particular game compared to overcoming strategy fusion. A useful observation from this work is that a strong level of play can be achieved in some imperfect information games without modelling inference or bluffing.

Hypothesis 4: Knowledge captured from simulations be exploited to create better MCTS policies The second main outcome of the project is the *Information Capture and Re-Use* (ICARUS) framework. The ICARUS framework enables modifications to MCTS (and ISMCTS) algorithms to be explicitly defined which facilitates the comparison and combination of different enhancements. Enhancements to MCTS often "supercharge" the algorithm by making each iteration more expensive, but each iteration more powerful. Successful enhancements improve the performance of MCTS when a fixed time budget is used for decisions (rather than a fixed number of iterations). ICARUS has been used to test combinations of both existing and new enhancements to investigate which are the most effective. In particular the enhancement NAST-2 [31] appears to be a good replacement for the default random simulation policy of the UCT algorithm, improving performance in a wide variety of domains. This results in an elegant formulation of MCTS in which the same algorithm (UCB1) is used to select every action in a simulated game.

1.3.1 Additional Project Outcomes

Finally there are several outcomes of the project not discussed in this thesis:

- Firstly the ISMCTS has been integrated into a commercial game Spades by AI Factory [10]. In order to satisfy the requirement of a mobile platform, work was done to modify MCTS run run within a fixed memory limit (but unlimited tie constraint). This work forms the basis of a submitted paper (listed in Table 1.2). The unmodified ISMCTS algorithm outperformed the existing heuristic based AI in the game. Based upon feedback

from users components of the heuristic AI were integrated with ISMCTS to make the behaviour of ISMCTS more intuitive. These modifications did not improve the strength of ISMCTS, but make the behaviour more realistic (particularly in situations ISMCTS determined the game was won or lost). Since integrating ISMCTS in the the product, it has risen to become the most popular Spades application on the android market place.

- As part of this project an entry to the 2012-13 PTSP competitions was developed using MCTS with macro actions, which won each competition [24, 25] as well as the multi-objective PTSP competition [32]. The controller combines a high level planner responsible for long term decisions and a low level planner, which uses MCTS to determine which granular inputs to make. This architecture is highly effective for the PTSP domain and it is a subject of ongoing work to adapt the architecture to other domains.

Whilst working on this project I co-authored 9 publications in peer reviewed journals and conference proceedings (listed in Table 1.1) with a further 5 in preparation or under review (listed in Table 1.2).

Publication	**Contributions**	**Location**
E. J. Powley, Peter. I. Cowling, and D. Whitehouse, "Information capture and reuse strategies in Monte Carlo Tree Search, with applications to games of hidden information", Artificial Intelligence, vol. 217, 2014, pp 92-116.	Contributed many ideas to the design of the ICARUS framework and the notation. Implementation and testing of the different enhancements. Initial design of the EPIC algorithm (for Dou Di Zhu) and the use of UCB for simulation policy. Analysis and interpretation of results. Significant writing contributions.	Chapter 8
P. I. Cowling, S. Devlin, E. J. Powley, D. Whitehouse, J. Rollason, "Player Preference and Style in a Leading Mobile Card Game", to appear in *IEEE Trans. Comp. Intell. AI Games* 2014.	Design and implementation of experiments to measure the effect of win rate on player retention. Evaluation of star rating as a grading of AI difficulty. Analysis and interpretation of results. Significant writing contributions.	Outside Thesis
D. Perez, E. J. Powley, D. Whitehouse, S. Samothrakis, S. M. Lucas and P. I. Cowling, "The 2013 Multi-Objective Physical Travelling Salesman Problem Competition", *Proc. IEEE Congr. on Evol. Comput.*, Beijing, China, 2014, pp 2314-2321.	Contributions to writing the section describing the Purofmovio controller.	Outside Thesis
D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, S. M. Lucas "Solving the Physical Travelling Salesman Problem: tree search and macro-actions" *IEEE Trans. Comp. Intell. AI Games.*	Design of the Purofvio controller and investigation of macro-action techniques. Contributions to the design of experiments and analysis and interpretation of results. Significant writing contributions.	Outside Thesis

Publication	Contributions	Location
D. Whitehouse, P. I. Cowling, E. J. Powley, J. Rollason, "Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game" *Proc 9th AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, Boston, MA, 2013	Contributions to the implementation of ISMCTS in Spades and porting to the AI Factory game engine. Design, analysis and implementation of enhancements. Modifications to the implementation based on feedback from the developer and customers. Significant writing contributions.	Outside Thesis
E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with Macro-Actions and Heuristic Route Planning for the Multiobjective Physical Travelling Salesman Problem" *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, Canada, 2013, pp 73-80.	Ideas contributed to the design of modifications to the controller for the multi-objective problem. Modifications to CMA-ES implementation for multiple objective parameter tuning. Significant writing contributions.	Outside Thesis
E. J. Powley, D. Whitehouse, and P. I. Cowling, "Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search" *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, Canada, 2013, pp. 81-88.	Implementation of the enhancements used in experiments. Design of experiments and analysis and interpretation of results. Design of the UCB simulation policy technique. Significant writing contributions.	Chapter 8
E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with macro–actions and heuristic route planning for the Physical Travelling Salesman Problem", *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp 234-241.	Participated in the design and implementation of the controller. Design and implementation of the steering component. Contributed to testing and tuning of the controller leading up to the competition. Significant writing contributions.	Outside Thesis
P. I. Cowling, E. J. Powley, D. Whitehouse, "Information Set Monte Carlo Tree Search", *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 2, pp. 120-143, 2012.	Lead the development and implementation of the ISMCTS algorithms. Design and testing of the LOTR:C implementation. Analysis of the effect on branching factor on ISMCTS for Dou Di Zhu. Significant writing contributions including to the development of notation.	Chapters 5 and 6
C. Browne, E. J. 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", *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 1, pp. 1-43, 2012.	Collecting and organising MCTS publications. Initial writing of the MCTS enhancements chapters as well as writing contributions to other chapters.	Outside Thesis

Publication	Contributions	Location
D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu", *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 87–94.	Design and testing of the information set UCT algorithm. Experiments comparing determinzed UCT and information set UCT. Analysis and interpretation of results. Significant writing contributions.	Chapter 5
E. J. Powley, D. Whitehouse, and P. I. Cowling, "Determinization in Monte–Carlo Tree Search for the card game Dou Di Zhu", *Proc. Artif. Intell. Simul. Behav.*, York, United Kingdom, 2011, pp. 17-24.	Design and implementation of the Dou Di Zhu game rules. Literature review on search in hidden information games. Contributed to the design of experiments. Significant writing contributions.	Chapter 5

Table 1.1: A list of all accepted publications written as part of the hidden information and uncertainty strand of the UCT for games and beyond project, along with a summary of my personal contributions to each piece of work

Publication	Contributions	Location
P. I. Cowling, E. J. Powley and D. Whitehouse, "Memory Bounded Monte Carlo Tree Search", unsubmitted.	Game implementations and testing and running experiments. Contributed ideas which went into the new algorithm and the selection of benchmark methods. Analysis and interpretation of results. Significant writing contributions.	Outside Thesis
D. Whitehouse, E. J. Powley, P. I. Cowling, "Inference and bluffing in Monte Carlo Tree Search", unsubmitted.	Game implementations for Scotland Yard and Saboteur. Contributed many ideas to the design of self-determinizing MCTS and MCTS based opponent modelling. Analysis and interpretation of results. Significant writing contributions.	Chapter 7

Table 1.2: A list of currently unpublished articles written as part of the hidden information and uncertainty strand of the UCT for games and beyond project, along with a summary of my personal contributions to each piece of work

1.4 Thesis Overview

In this section an overview of the structure of this thesis is presented. Firstly, chapter 2 introduces notation used throughout this thesis (adapted from the publications of this work) as well as definitions which are used in subsequent chapters. Chapter 3 surveys the literature in more depth and introduces the ideas from game theory, artificial intelligence and MCTS which the new results in this thesis are built upon as well describing other comparable techniques. Chapter 4 describes the software framework which was developed for

testing the new ideas in this thesis and implementing the domains which are used in experiments. Additionally the rules of each domain is described along with a summary of other research on AI in those domains. Chapter 5 covers the integration of MCTS with the existing determinization based Perfect Information Monte Carlo (PIMC) approach, discusses the flaws of PIMC and how they can be overcome by using information set trees and ISMCTS. Chapter 6 shows how partial observability can be handled with ISMCTS by introducing the MO-ISMCTS algorithm as well as experiments showing the limitations of this approach. Chapter 7 investigates how bluffing and inference can be performed using ISMCTS, overcoming the problems with MOSICMTS. This leads towards a more theoretically sound version of the ISMCTS at the cost of large computational requirements due grouping fewer information sets together Inference is introduced to ISMCTS, using ISMCTS as an opponent model. Finally different methods of inducing bluffing behaviour and reducing exploitability are tested. Chapter 8 introduces the ICARUS framework and shows how existing enhancements can be described in the framework and a new class of enhancements called EPIC is introduced. Experiments compares different combinations of both new and existing enhancements as well as testing the most effective selection methods for simulation policies. Finally Chapter 9 concludes the research and discusses the significance of the new contributions of this work as well as identifying interesting directions for future work.

Chapter 2

Notation and Definitions

This chapter introduces the definitions and important concepts that are referred to in subsequent chapters as well as the notation used to describe a game. This notation is an extension of the notation used in the original paper describing the ISMCTS algorithm [27]. The structure of this chapter is as follows. Firstly Section 2.1 informally describes the concept of games and game trees as different sources of hidden information and uncertainty. Section 2.2 precisely defines the notion of a game and introduces the notation used throughout this thesis. Section 2.3 introduces the idea of a policy for a game and several important classes of policies. Finally Section 2.4 describes the methodology used to design the majority of the the experiments presented in this thesis.

2.1 Games with Hidden Information and Uncertainty

The class of games which are studied in this work are physical board and card games which are usually played with a group of human players. Many modern games such as collectable card games and eurogames feature a huge variety of game mechanics and often high levels of complexity. In addition physical games and other turn based games are played using a computer, with a mix of human players and computer agents. This thesis is concerned with the development of computer agents for playing games, which can produce engaging play for human players and can run on the hardware constraints of a commercial game.

Games can be classified according to the following properties:

- Zero-sum: Whether the reward to all players sums to zero (or equivalently some constant value) for all terminal states.
- Information: Whether the state of the game is fully or partially observable to the players.

- Determinism: Whether or not taking the same action from the same state always results in the same outcome.
- Sequential: Whether actions are applied by players sequentially or simultaneously.
- Discrete: Whether actions are discrete or applied in real-time.

A game can be represented as a digraph, where nodes correspond to states of the game and edges correspond to transitions between states. A game begins in some initial state determined by the rules of the game and in each state a player to act chooses an action which encodes a transition from that state. This process continued until a terminal state is reached. In each state each player may be awarded some utility value and typically each player aims to maximise their cumulative utility over the course of the game. For simplicity it can be assumed that the awarding of utility values is deferred until a terminal state. If the sum of utility values for all players adds up to 0 a game is known as a *zero-sum* game. In the 2-player case this means that one player's reward is the negative of the other player's, i.e. what one player wins, the other must lose.

A game is played by one or more game players, who decide which actions to take given their observation of the state of a game. Players of a game do not always observe the exact state of a game, rather players know that the current state must belong to a set of possible states known as an information set (determined by the rules of a game). Furthermore actions taken by players encode a set of state transitions, with one transition for each possible state within the information set. Which transition occurs when the action is applied depends on the actual state of the game. In games where information sets are singletons which contain only the exact state of the game, this game is said to have *perfect information*, otherwise a game is said to have *imperfect information*.

In some games state transitions are non-deterministic, in other words taking the same action from the same state can have different outcomes. Games with this property are known as *stochastic games*, or games with *uncertainty*. Games with no uncertainty are known as *deterministic* games. One way of modelling uncertainty is by inserting an extra state in which no player takes an action, rather the game chooses a random transition from that state. This ensures that actions always encode a single transition from a particular state. In the implementation of games in this thesis, the outcomes of random events are chosen by an environment player "0" (the actual players being numbered $1, 2, 3 \ldots$) that always chooses randomly.

There are several well studied classes of games, for example *Markov decision processes* (MDP) which are single player games of perfect information with uncertainty and *Partially observable Markov decision processes* (POMDP) which are MDPs with imperfect information. Another well studied class of games is *combinatorial games*, which are two player, deterministic, zero sum, perfect information games with a finite number of states. The work in this thesis is concerned with classes of games which relax these constraints. In other words games with any number of players, which may not be zero-sum as well as having

imperfect information and uncertainty. In particular several sources of imperfect information and uncertainty are considered:

- Hidden Information: Games in which some information is not visible to some players and information sets contain many states. For example in a card game, a player may not know the identities of cards in the hands of other players.
- Opponent Strategy: Often the effectiveness of a strategy may depend on the strategy chosen by another player and the strategy of other players may be unknown, or unpredictable. For example in a game of rock, paper, scissors always choosing rock will beat a player always choosing scissors, but lose to a player always choosing paper.
- Partially Observable Actions: Players may not always observe the exact action taken by another player, but rather a set of possible actions. For example a player may observe another player look through a shuffled deck and draw a card but not observe the identity of the drawn card.
- Simultaneous Actions: Games in which players make decisions simultaneously, for example the game rock, paper scissors. In addition some games may have both sequential and simultaneous decisions. Simultaneous actions can be modelled as sequential partially observable actions.
- Stochastic: Some events in a game may be randomly determined. For example dice rolls or deck shuffling. Stochastic outcomes can be considered hidden information, for example if all dice rolls are fixed in advance but hidden from players.

Finally, an imperfect information game can be converted to a perfect information game through a process known as *determinization*. A determinization is a state from an information set which is fixed as the true state of the game, then the game is played out assuming everything is fully observable. For example in a card game this could be equivalent to playing with all cards face up. Crucially, the determinization process removes all sources of hidden information and uncertainty, for example by fixing all future stochastic outcomes and making this information visible to all players.

It should also be noted that partially observable actions are often regarded as hidden information, since the result of any action is that players transition from one information set to another. However in this work, observation of action histories is used to generate and distinguish between information sets so it is important to take account each players observations of a transition. By taking this approach, the ISMCTS algorithm can share information across information sets and avoid to actually storing information sets (in order to distinguish between them). The motivation for using this representation is that there is typically a small amount of information contained in a move compared to an information set, so choosing the correct edge in a tree (corresponding to an observation of a move) is more computationally efficient than finding the correct information set.

This enables a highly efficient implementation of the ISMCTS algorithm, but comes at the cost of a significant increase in complexity of implementation. In particular, care must be taken when information is revealed or hidden since this can only be achieved through actions (corresponding to transitions in the MCTS search tree). This is implemented by having the environment player make extra actions which reveal (or conceal information) to players where necessary. This would not be needed if a representation based on information sets was used.

2.2 Game Notation

In this section a game is precisely defined and game notation introduced. Firstly a few useful notations need to be defined. For a set X, a sequence over X is written as $\langle x_1, \ldots, x_n \rangle$ for $x_i \in X$. The empty sequence is denoted $\langle\rangle$. The set of all sequences over X is denoted X^*. The concatenation of two sequences $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ and $\boldsymbol{y} = \langle y_1, \ldots, y_m \rangle$ is $\boldsymbol{x} + \boldsymbol{y} = \langle x_1, \ldots, x_n, y_1, \ldots, y_n \rangle$. The concatenation operator can be used for prepending or appending single elements to a sequence, for example $\boldsymbol{x} + x_{n+1} = \langle x_1, \ldots, x_n, x_{n+1} \rangle$ for $x_{n+1} \in X$. Let X be a set and let $\sim$ be an equivalence relation on X. Then $[x]^\sim$ is the $\sim$-class of $x \in X$, and $X/\sim$ is the set of all $\sim$-classes.

Now the terminology and notation for games is introduced. The notation was introduced in [27] and is extended here. More detail on the concepts underlying the definition of a game can be found in [33] or other standard textbooks on game theory. A *game* is defined on a directed graph (S, Λ). The nodes in S are called *states* of the game; the leaf nodes are called *terminal states*, and the other nodes *nonterminal states*. Λ is the set of state transitions. A game has a positive number κ of players, numbered $1, 2, \ldots, \kappa$. There is also an *environment player* numbered 0. Each state s has associated with it a number $\rho(s) \in \{0, \ldots, \kappa\}$, the *player about to act*. Each terminal state s_T has associated with it a vector $\mu(s_T) \in \mathbb{R}^\kappa$, the *reward vector*.

The game is played as follows. At time $t = 0$ the game begins in the *initial state* s_0. At time $t = 0, 1, 2, \ldots$, if state s_t is nonterminal, player $\rho(s_t)$ chooses an edge $(s_t, s_{t+1}) \in \Lambda$ and the game transitions through that edge to state s_{t+1}. This continues until a time $t = T$ when s_T is terminal. At this point, each player receives a reward equal to the relevant entry in the vector $\mu(s_T)$, and the game ends.

Players typically do not choose edges directly, but choose *actions*. Actions are equivalence classes of edges, with the restriction that two edges starting from the same node cannot be in the same action. It is also required that all edges in an action have the same player about to act in their start nodes. Actions capture the notion that different edges from different states can be in some sense equivalent: for example, an action may consist of all edges leading from a state where player 1 holds an ace (and some other cards) to the state where player 1 has just played the ace; such an action would be labelled "player 1 plays an ace". The set of actions from a state s is denoted $A(s)$; this is simply the set of action classes restricted to edges outgoing from s.

The *transition function* f (see Definition 3 below) maps a (state, action) pair (s_t, a) to a resulting state s_{t+1}, by choosing an edge $(s_t, s_{t+1}) \in a$. Note that the domain of f does not include all (state, action) pairs, as not all actions are available in each state.

A *policy* for player i maps each state s with $\rho(s) = i$ to a probability distribution over $A(s)$. This distribution specifies how likely the player is to choose each action from that state. One way of stating the fundamental problem of game AI (and indeed of game theory) is as finding the policy that leads to the highest expected reward, given that all other players are trying to do the same. The exception here is the environment player, whose policy is fixed as part of the game definition and specifies the probabilities of outcomes for chance events.

This thesis studies games of *imperfect information*. In these games, each player partitions the state set S into *information sets*. Note that in general, each player's partitioning is different. See Figure 5.10 (page 74) for example. The players do not observe the actual state of the game, but rather the information set containing the actual state. Essentially, the states in a player's information set are indistinguishable from that player's point of view. In particular this means that the player's choices of actions must be predicated on information sets, not on states.

This thesis also studies games with *partially observable moves*. Here each player further partitions the actions into *moves*. It is required that the partitions for a player's own actions are singletons, i.e. that players can fully observe their own moves. When a player plays an action, the other players do not observe that action directly but observe the move to which the action belongs. The set of moves from player i's point of view from a state s is denoted $M_i(s)$, and is the set of move classes restricted to edges outgoing from s. In the case where $\rho(s) = i$, let $M_{\rho(s)}(s) = A(s)$.

It is required that two edges leading from states in a player i information set and contained within the same move from player i's point of view must lead to states in the same player i information set. In other words, all available information about the game can be gathered by observing moves, without the need to directly observe information sets. This also allows a transition function on (information set, move) pairs to be well defined. This property can be achieved if necessary by addition of *environment states*, in which the environment player has exactly one action available; this action may depend upon (and thus provide an observation of) the actual state, but different states in the same information set from the point of view of some other player may have different actions available. Environment states are necessary to enable AI player to distinguish between states in which different sets of information have been revealed or hidden, without having to have an game specific encoding of information in a game.

In summary, the following definitions describe a game:

Definition 1. A *game of imperfect information* is a 9-tuple

$$\Gamma = (S, \Lambda, s_0, \kappa, \mu, \rho, \pi_0, (\sim_1, \ldots, \sim_\kappa), (\smile_1, \ldots, \smile_\kappa)) \tag{2.1}$$

where:

- (S, Λ) is a finite nonempty directed graph, with S the set of *states* and Λ the set of *state transitions*;
- $s_0 \in S$ is the *initial state*;
- $\kappa \in \mathbb{N}$ is the *number of players*;
- $\mu : S_T \to \mathbb{R}^\kappa$ is the *utility function*, where $S_T \subseteq S$ is the set of leaf nodes (*terminal states*);
- $\rho : S \to \{0, 1, \ldots, \kappa\}$ defines the *player about to act* in each state;
- $\pi_0 : \Lambda_0 \to [0, 1]$, where $\Lambda_0 = \{(r, s) \in \Lambda : \rho(r) = 0\}$ and $\sum_{\{s \,:\, (r,s)\in\Lambda_0\}} \pi_0(r, s) = 1$ is the *environment policy*;
- $\sim_i$ is an equivalence relation on S, whose classes are player i's *information sets*;
- $\backsim_i$ is an equivalence relation on Λ, whose classes are *moves* as observed by player i. Classes consisting of edges (s, u) with $\rho(s) = i$ are also known as player i's *actions*.

In a game of imperfect information, players do not observe the current state but observe the information set that contains it. Likewise they do not observe state transitions or actions but moves. (Note the distinction between state transitions, actions and moves: a player chooses an *action*, which induces a *state transition*, and the other players observe a *move*.) An information set consists of all states that are indistinguishable from the player's point of view; a move consists of all actions that are indistinguishable from the player's point of view. Thus a player's choices of action can depend only on the information sets and moves that he observes, not on the underlying states and actions.

Definition 2. Consider a game Γ, a state s and a player i. The set of *legal moves* from state s from player i's point of view is

$$M_i(s) = \{[(s, u)]^{\backsim_i} \;:\; (s, u) \in \Lambda\} \,. \tag{2.2}$$

The set of *all moves from player i's point of view* is the set of all moves legal in at least one state:

$$M_i = \Lambda/\backsim_i = \bigcup_{s\in S} M_i(s) \,. \tag{2.3}$$

The set of *all moves* is the set of all moves from all players' points of view:

$$M = \bigcup_{i=1,\ldots,\kappa} M_i \,. \tag{2.4}$$

The set of *legal actions* from s is

$$A(s) = M_{\rho(s)}(s) \,, \tag{2.5}$$

i.e. the set of legal moves from the point of view of the player about to act. The set of *all actions* is the set of all actions legal in at least one state:

$$A = \bigcup_{s \in S} A(s)\,. \tag{2.6}$$

Definition 3. Consider a game Γ. Let $B = \{(s, a) \ : \ s \in S, a \in A(s)\}$, the set of all pairs of states and their legal actions. The *transition function* for Γ is the function $f : B \to S$ such that given $s \in S$, $\forall a \in A(s)$, $(s, s') \in a \Rightarrow f(s, a) = s'$. In other words: $f(s, a)$ is the state reached by starting from s and traversing the edge corresponding to a; $f(s, a)$ is the state resulting from performing action a in state s.

Definition 4. An *action history*[1] from state s is a sequence of actions $\langle a_1, \ldots, a_n \rangle \in A^*$, such that

$$a_1 \in A(s) \tag{2.7}$$
$$a_2 \in A(f(s, a_1)) \tag{2.8}$$
$$a_3 \in A(f(f(s, a_1), a_2)) \tag{2.9}$$
$$\vdots \tag{2.10}$$
$$a_n \in A(f(\ldots(f(s, a_1), \ldots), a_{n-1}))\,. \tag{2.11}$$

Denote the set of all action histories from s by $H(s)$. Extend the transition function f to operate on action histories by defining

$$f(s, \langle\rangle) = s \tag{2.12}$$
$$f(s, \langle a_1, \ldots, a_n \rangle) = f(f(s, \langle a_1, \ldots, a_{n-1} \rangle), a_n)\,. \tag{2.13}$$

An action history $\boldsymbol{h}$ is *terminal* if $f(s, \boldsymbol{h})$ is a terminal state. Denote the set of terminal action histories from s by $H(s)$.

Definition 5. A *move history* for player i from state s is a sequence of moves from player i's point of view, $\langle [a_1]^{\frown_i}, \ldots, [a_n]^{\frown_i} \rangle \in M_i^*$, where $\langle a_1, \ldots, a_n \rangle$ is an action history from s. Denote the set of all move histories for player i from s by $H_i^{\frown}(s)$, and the set of all move histories for all players by $H^{\frown}(s)$. If $\boldsymbol{h} = \langle a_1, \ldots, a_n \rangle$ is an action history then the corresponding move history from player i's point of view is denoted $[\boldsymbol{h}]^{\frown_i}$. Let $\rho = \rho(f(s, \langle a_1, \ldots, a_{n-1} \rangle))$, so ρ is the player who played the last action a_n in the history. Then the move history from player ρ's point of view is denoted by omission of the player number, i.e. $[\boldsymbol{h}]^{\frown}$.

Definition 6. A *game tree* with root node r is formed from the action histories $H(s_r)$ of a root state s_r. Each node in a game tree is labelled with a state $s \in S$ and a history $h \in H(s_r)$ in such that $f(s_r, h) = s$. The root node is labelled

[1] Note that a history from state s begins, not ends, at state s. If s is considered to be the current point in time, a "history" could more correctly be called a "future".

with state s_r and history $h = \langle\rangle$. The edge which connects a node n with its parent node n is labelled with the last action in the history of n. In a *complete* game tree, leaf nodes are members of $H(S_r)$. If a game tree is note complete, it is said to be a *partial* game tree. Game trees have the following properties:

- Each node in the tree is labelled with exactly one state, but one state can be the label of several nodes.
- The history of a node is precisely the sequence of moves or actions that label the edges from the root to that node.
- The *depth* of the tree is the length of the longest history.
- The out-degree of a node is the *branching factor* of its state.

2.3 Solution concepts for games

Players of a game make decisions according to a policy π which maps every information set where that player is to act to a probability distribution over the set of legal actions.

Definition 7. Let π_j denote the policy of a player j where $\pi_j : M_j(s) \mapsto [0, 1]$ and $\Sigma_{a \in M_j(s)} \pi_j(a) = 1)$, $\forall s \in S$ such that $\rho(s) = j$

Definition 8. If $\pi_j(s) \in \{0, 1\}$ $\forall s \in S$ with $\rho(s) = j$ then π_j is a *pure policy* otherwise π_j is a *mixed policy*

If a player has a pure policy, then all action choices are deterministic and the policy will always choose the same action from the same state. In combinatorial games, there is always at least one pure policy which is optimal [33] known as a minimax strategy (there may be more than one, and there may be mixed policies that are optimal too). Since the policy is deterministic the outcome of the a game will always be the same if both players are using an optimal policy. This means that combinatorial games can have two possible solutions since they are zero sum, one in which one player wins and another loses (the rewards for each player are not equal) and another in which the game is a draw (the rewards for both players are equal to zero). The largest combinatorial game which has been solved in this way is Checkers, which is a draw with optimal play [34].

For games with imperfect information, pure policies are often not optimal and therefore the outcome of the game may be not deterministic. When comparing policies in imperfect information games, it is therefore natural to consider the expected reward of a policy. For games with multiple players, the expected reward of a policy will depend on the set of policies chosen for each player. The aim of each player is typically to find a policy which maximises their expected reward. However the expected reward of a policy depends on the policy chosen by other players and it may be the case that a policy may be superior to some policies and inferior to others. Therefore players may instead looks for policies

that are robust against all possible policies chosen by other players, which leads to the notion of an equilibrium of policies.

Definition 9. For a game, a set of policies forms an ϵ-*equilibrium* if no player could gain more than ϵ in expected reward by unilaterally changing policy. If $\epsilon = 0$ then this is known as a *Nash-Equilibrium*.

As previously discussed, in combinatorial games there always exists a pure policy Nash-equilibirum [33] (known as the minimax solution). If all players are playing a pure strategy this is an ϵ-equilibrium where the ϵ for this set of strategies is the most any single player could gain in expected reward by playing a different (possibly mixed) strategy. The best counter to a pure strategy can be found by enumerating all counter strategies and choosing the one with the highest reward (or playing a mixture of the counter strategies which yield the same maximal reward).

Nash equilibria can be calculated precisely using iterated elimination of dominated strategies [33] or linear programming [35], or approximated using techniques such as counterfactual regret minimization [36, 37]. However computing a Nash-Equilibirum is infeasible for most real games. Game theory often assumes that all players are rational; that is, they always act to maximise their own reward. Under this assumption when playing a game the policies chosen by player should form a Nash equilibrium. However this assumption is often not true in practice given the difficulty of computing Nash equilibria for nontrivial games. If one player has already deviated from a Nash equilibrium then the other players can exploit this by deviating themselves.

2.4 Statistical Analysis of Experiments

The experiments presented in this thesis mostly consist of a game where the players are controlled using a set of AI methods. Since the policies produced by the AI techniques are often mixed and most of the games are stochastic in nature, it is natural to compare algorithms using the probability each algorithm will win a game, which can be estimated by averaging the number of games won across a large number of repeated games. That is, for a fixed set of AI methods playing a particular game, each algorithm has a probability p of winning the game.

For each algorithm, a game can be considered a Bernoulli trial where the algorithm has probability p to win the game and probability $1-p$ of not winning the game. Therefore the number of games won by an algorithm in a series of repeated games is binomially distributed. The uncertainty in the estimation of p can be calculated using Clopper-Pearson intervals [38]. Where error bars are displayed on figures in this thesis they are 95% Clopper-Pearson intervals.

Chapter 3

Literature Review

The main topic of this thesis is the application of Monte Carlo Tree Search to imperfect information games. This chapter surveys the existing work that the new ideas in this thesis are built upon. Firstly the multi-armed bandit problem is introduced in Section 3.1, which is a key component of the MCTS algorithm. Next Section 3.2 presents an overview of existing work on MCTS including with how the MCTS family of algorithms is defined and what important variants and applications of the algorithm exist. Finally Section 3.3 presents a summary of existing work applying tree search algorithms to imperfect information games.

3.1 Multi-Armed Bandits

In the multi-armed bandit problem, a gambler has a choice of several bandit machines each of which has a different probability of paying out. The goal of the problem is to balance spending time evaluating which machine has the highest probability with spending as much time as possible playing the best machine. This is an example of the exploration versus exploitation dilemma. In general a multi-armed bandit has a finite set of arms, each with an unknown reward distribution and the problem is to define a policy which takes as input all rewards received on all previous trials and determine which arm should be selected next.

Regret is a term used to describe the loss incurred by playing sub-optimal arms, which is the difference between the reward received and the expected reward from the optimal arm. The cumulative regret is the sum of the regret incurred on all previous trials. The standard way of evaluating the strength of a policy for the multi-armed bandit problem is determine the expected value of the cumulative regret after a fixed number of trials. In the case of the UCB policy [39] the expected cumulative regret grows logarithmically with the number of successive trials.

The multi-armed bandit problem has applications in many areas and interest the problem pre-dates the application in MCTS, where the selection of an action

in a game tree is be treated as a multi armed bandit problem. In this case the arms correspond to the different actions and the rewards are generated by simulating the outcome of the game. The UCT algorithm [2] builds a partial game tree iteratively and the UCB1 algorithm [39] us used to select actions within the tree (and Monte-Carlo simulation is used to select actions from states outside the tree). For a game of perfect information the optimal action will be the action with the highest minimax value and the UCT algorithm converges to this value as the number of iterations increases.

3.1.1 k-Armed Multi-Armed Bandit Problem

The bandit problem previously described is also known as the k-Armed Multi-Armed Bandit Problem to distinguish it from other variants of the multi armed bandit problem. The k-Armed Multi-Armed Bandit Problem is defined as follows:

Definition 10. A *k-Armed Multi-Armed Bandit Problem* consists of random variables $X_{i,n}$ for $1 \leq i \leq k$ and $n \geq 1$. This corresponds to the arms on k gambling machines where i denotes the index of each arm and n denotes the reward received on the n^{th} play of an arm. In addition random variables $X_{i,n}$ satisfy the following two properties:

- Random Variables $X_{i,n}$ for $n \geq 0$ are independent and identically distributed according to some distribution with expectation $\mathbb{E}[X_{i,n}] = \mu_i$.
- Random Variables $X_{i,s}, X_{j,t}$ are independent but not necessarily identically distributed for $1 \leq i < j \leq k$ and $s, t \geq 1$.

i.e. the arms may have different reward distributions, but (in this version of the multi-armed bandit problem) the distribution for a given arm does not change over time The term *trial* is used to describe the process of selecting an arm to play and then receiving a reward. The number of times an arm i was played during the first n trials is denoted by $T_i(n)$.

3.1.2 Bandit Algorithms

There exists many different policies for the multi-armed bandit problem, also known as bandit algorithms. A bandit algorithm decides which arm to pull on a given iteration, given the history of previous pulls and rewards. In this work UCB1 [39] is the bandit algorithm used for action selection in MCTS which select the arm which maximises

$$\bar{X}_{i,n} + C\sqrt{\frac{\log n}{T_i(n)}} \tag{3.1}$$

where $\bar{X}_{i,n}$ is the average reward obtained after $T_i(n)$ trials of arm i (out of n trials in total) and C is a numerical constant. The UCB1-Tuned algorithm is also used in some MCTS applications [40, 15], which removed the need to

tune constant C and often leads to better performance. Additionally the EXP3 algorithm [41] has been applied to simultaneous action selection [42] and is used for simultaneous action selection in this work. The EXP3 algorithm was designed for a variant of the multi armed bandit problem where the rewards are manipulated by an adversary, thus more accurately models the problem of choosing simultaneous actions (where the action chosen by the opponent alters the rewards).

3.2 Monte Carlo Tree Search

Monte Carlo Tree Search is a family of tree search algorithms which first emerged in 2006. The idea of using monto carlo simulations in game AI was not new, for example Abramson [18] demonstrated that many random games could be used to estimate the game-theoretic value of a move. Flat Monte-Carlo Search is a search algorithm which uses random games to evaluate moves and has been applied to Go [43]. The innovation of *Monte Carlo Tree Search* (MCTS) was to use random simulations as an estimator for the value of a move in combination with a search tree to guide simulations and balance the exploitation of promising moves with the exploration of untried moves. This idea was proposed by Coulom [4] as well as Kocsis and Szepesvri [2] which introduced the UCT algorithm. Almost immediately MCTS led to great advances in the field of computer Go [44, 45] and all of the top computer Go programs are now MCTS based [7]. Additionally there has been a lot of research into the application of MCTS to other domains [1] leading to advances in many different problems.

MCTS has mostly been applied to games of perfect information. This is unsurprising given that the UCT algorithm converges to the minimax solution to these games. MCTS has produced the strongest programs for games such as Go [7] and Hex [46] and has shown to be highly scalable in parallel search applications [47]. MCTS has also been applied to imperfect information games in combination with determinization approaches such as perfect information monte carlo search in domains such as Solitaire [48], Skat [49], Dou Di Zhu [50] and Magic: The Gathering [51, 52]

3.2.1 Monte Carlo Tree Search Algorithms

There are many different variations of MCTS algorithms [1], however all MCTS algorithms follow the same general structure. MCTS performs a large number of simulated games and builds a partial game tree adding one node on each iteration. MCTS makes a decision based on the outcomes of the simulations which are stored in this tree. Each simulated game is referred to as an *iteration* of the MCTS algorithm and consists of the following four steps:

1. Selection: Choose moves whilst balancing the exploitation of good moves with the exploration of untried moves until a state is reached which is not represented in the MCTS tree.

2. Expansion: Add a new node to the MCTS tree corresponding to the unvisited state encountered during selection.

3. Simulation: Play out the rest of the game using a randomised policy.

4. Backpropagation: Update nodes in the tree with the outcome of the game in the simulation step.

Precisely how each step is performed varies between MCTS algorithms but the major components of an MCTS algorithm are the policies used for selection and simulation. In the case of the UCT algorithm [2] the selection policy is UCB1 [39] and the simulation policy is purely random. In Chapter 8 a general framework for MCTS algorithms is introduced which makes this structure more formal whilst allowing all existing MCTS algorithms to be defined within the framework.

3.2.2 Properties of Monte Carlo Tree Search

MCTS has several useful properties in comparison to other search algorithms. Firstly MCTS can be applied to any problem which is modelled by a decision tree and can be applied without any domain knowledge (although inclusion of domain knowledge provides benefits). Therefore MCTS can be characterized as *aheuristic*. This is in contrast to the popular approach of minimax search, where the quality of play depends significantly on the heuristic used to evaluate the probable outcome non-leaf nodes. In games such as Chess where reliable heuristics have emerged after decades of research, minimax performs well. In cases such as Go, however, where branching factors are orders of magnitude larger and useful heuristics are difficult to find (at least for minimax), the performance of minimax degrades significantly. MCTS has been highly successful in the domain of *General Game Playing* (GGP) [8] where the rules of games used to evaluate techniques are not known in advance.

MCTS is also an *anytime* algorithm, in that it can be stopped at any point and return a decision. This is in contrast to algorithms such as minimax search which must complete a computation before a decision can be made. It is possible to have an anytime version of minimax search using iterative deepening. Nevertheless, as an entire ply is added to the tree on each iteration, the granularity of progress is much coarser, even if techniques such as α-β pruning can be used to improve efficiency.

Finally MCTS performs an *asymmetric* search, rather than a fixed depth search. This enables MCTS to spend more time searching plausible future lines of play in great depth and less time searching areas of the tree with sub-optimal moves.

There are also weaknesses to MCTS algorithms. MCTS has been demonstrated to be inferior to minimax search in domains where good heuristics are known [53] (but heuristics can be used with MCTS too). In many implementations of MCTS a suite of enhancements and domain knowledge needed to be

used in order to achieve good results [1]. However, the integration of heuristic knowledge with MCTS is also not always guaranteed to lead to stronger play and can even be detrimental [54]. Also many bandit algorithms used for selection in MCTS have parameters which must be tuned for each domain (although good default values exist), although typically MCTS algorithms have a small number of tunable parameters compared to the sophisticated evaluation functions developed for use with minimax search.

3.2.3 Enhancements and Applications

There are many different enhancements for MCTS [1] and they can be divided into two classes. Firstly there are general purpose enhancements which improve the performance of MCTS without utilizing any domain specific knowledge and can potentially be used in any MCTS application. General purpose enhancements typically add complexity to the MCTS algorithm by modifying the tree selection (for example RAVE [54]) or simulation policies (for example MAST [55]). Any benefit of the enhancement must outweigh the reduced number of simulations that can be performed in a given amount of time. General purpose enhancements may only work in particular types of domains and be detrimental in others: despite not being tailored to a specific game, they may still rely on the game having certain characteristics (for example a notion of certain moves being good or bad regardless of the current state). There are also often additional tunable parameters introduced with general purpose enhancements. Enhancements to MCTS which provide methods for exploiting domain specific knowledge are also well studied, for example patterns in Go [44, 56, 7], Hex [46] and Othello [57]. However the work in this thesis is concerned with developing MCTS algorithms which can be applied to large classes of games, so these enhancements were not used.

One thing most of these methods have in common is that they alter how MCTS learns from simulated games and how this learned knowledge is used to influence the tree selection and simulation policies. The ICARUS framework introduced in Chapter 8 generalizes MCTS as an algorithm which captures information from simulated games and re-uses this information to build improved tree selection and simulation policies as more iterations are performed. Chapter 8 introduces several new enhancements and presents experiments studying variations of the following enhancements:

- The *all moves as first (AMAF)* heuristic was introduced by Brügmann [43] in the context of Monte Carlo methods for Go, and was first combined with MCTS by Gelly and Silver [54] and independently by Drake and Uurtamo [58]. The underlying idea is that the value of an action is somewhat independent of the time at which it is played. This time independence is particularly true for games with pieces that rarely or never move once played, such as Go and Hex. AMAF and its variants have proven highly successful in these [7, 21] and other similar games. AMAF updates statistics for each action in the playout not just at the point when that action

was played, but also at all earlier points when the action could legally have been played.

- *Move-average sampling technique (MAST)* was introduced by Finnsson and Björnsson [59] and used in their CadiaPlayer general game player [8]. The idea is to maintain average reward statistics for each action independently of where it occurs in the game tree, and use these statistics to bias the simulation policy.

- *Last good reply (LGR)* is a simulation policy introduced by Drake [60]. When playing a game, each action can be thought of as a reply to the opponent's previous move. If the replying player goes on to win the game, this provides some evidence that the reply was good. LGR records good replies from MCTS playouts; during simulation, if a good reply is recorded for the previous move then it is played deterministically. LGR has been shown to improve the performance of MCTS for Go [60, 61], Havannah [62] and General Game Playing [63].

Chapter 8.2.3 (page 147) surveys several additional enhancements and explains how they can be integrated into the ICARUS framework.

3.2.4 Simultaneous Moves

This section surveys existing work on the application of MCTS to games with simultaneous moves. Simultaneous moves are a source of imperfect information and occur in several of the games studied in this thesis. *Simultaneous moves* are a special case of imperfect information, in which each player independently chooses an action and these actions are applied at the same time. This can occur as a decision in a larger game, where some decisions are made sequentially and others simultaneously. Simultaneous actions occur in several of the games studied in this thesis, In particular in the game Lord of the Rings: The Confrontation [64] players take turns to move characters on a board. Occasionally these characters enter combat, which is decided by players simultaneously choosing from a set of cards to determine the outcome of combat.

Simultaneous moves can be modelled by having players choose their actions sequentially, but hiding their choices from the other players, until finally an environment action reveals the chosen actions and resolves their effects. With this in mind, any algorithm that can handle imperfect information in general can handle simultaneous moves in particular. However, some of the new algorithms in this thesis (particularly those not designed to handle partially observable moves) perform poorly using this model. Under a simple determinization approach, the first player is overly pessimistic (assuming the opponent can observe the chosen move and select the best response to it) while the second player is overly optimistic (assuming the first player's move is fixed at the point of the second player's decision, and thus determinizing it randomly).

When applying a tree search algorithm to a game with simultaneous actions, it is important that the response of one player is not tailored to the action chosen

by another. For this reason, a mechanism must be added to search algorithms to handle simultaneous moves. There are several existing techniques for making simultaneous decisions. If the rewards for each outcome are known then it is possible to compute a Nash equilibrium for the decision. When a simultaneous decision occurs within a sequential decision tree, the expected rewards for each outcome are not initially known. This means any tree search approach has to first calculate the rewards (dependent on the decisions made in each sub-tree) then use the rewards to produce a decision. One algorithm which can do this is CFR [36].

The UCT algorithm has been applied to the simultaneous move game rock-paper-scissors by Shafiei et al [65], using an approach where each player's choice of action is treated as a separate independent multi-armed bandit problem. In other words, instead of selecting player 1's move, descending the corresponding tree branch, and selecting player 2's move from the resulting child node, both moves are selected independently from the same node and the tree branch corresponding to the resulting pair of moves is descended. Shafiei et al [65] show that this approach finds mixed policies, though not necessarily Nash policies. More recently Lanctot et al [66] have introduced Online Outcome Sampling (OOS) which approaches a nash-equilibrium over time. This technique has been demonstrated to produce less exploitable strategies than MCTS in the simultaneous move game Goofspiel.

In this thesis the approach proposed by Teyaud and Flory [42] is used to handle simultaneous decisions. Teytaud and Flory [42] suggest a modification of the UCT algorithm, in which the UCB bandit algorithm is replaced by the EXP3 algorithm [41] at nodes with simultaneous moves only (i.e. UCB is still used elsewhere in the tree). As with the approach of Shafiei et al [65], simultaneous moves are chosen using independent bandits. The justification for using EXP3 rather than UCB is that the optimal policy at a simultaneous move node is often mixed; UCB is designed to converge to a pure policy, whereas EXP3 explicitly seeks a mixed policy. Teytaud and Flory [42] further strengthen this justification by comparing the playing strength of UCB versus EXP3 for the card game Urban Rivals, showing that EXP3 performs better and requires less tuning.

In EXP3, the probability of selecting an arm $a \in A$ is

$$p(a) = \frac{\gamma}{|A|} + \frac{1-\gamma}{\sum_{b \in A} e^{(s(b)-s(a))\eta}}, \tag{3.2}$$

where $s(a)$ is the sum of rewards from previously selecting arm a, each divided by the probability of selecting a on that trial, and η and γ are constant parameters. This equation is of a different form to that given by Auer et al [41], but is equivalent and more numerically stable.

Naturally the performance of EXP3 depends on the choice of coefficients. After [41, Corollary 4.2] let

$$\gamma = \min \left\{ 1, \sqrt{\frac{K \log K}{(e-1)n}} \right\} \tag{3.3}$$

and

$$\eta = \frac{\gamma}{K}\,, \tag{3.4}$$

where $K = |A|$ is the number of arms, n is the total number of trials, and e is the base of the natural logarithm.

3.2.5 Multiplayer Games

Much of the existing work on applications of MCTS has focused on 2 player zero-sum games However several of the domains studied in this thesis have more than 2 players. One approach to multiplayer games is to use the max^n idea [67], where a vector of rewards is given to each player and each player seeks to maximize their own reward. In the two-player zero-sum case, max^n is equivalent to minimax: maximising one's own reward is the same as minimising your opponent's reward. Max^n is not commonly used in minimax tree search as it is not compatible with alpha-beta pruning (although a limited version known as shallow alpha-beta pruning is possible [68] In MCTS there is no such reason not to use it. Indeed, max^n has been used with MCTS be several authors [69, 70] and is the approach used taken in this thesis. Cazenave [71] also considers how to handle coalitions between players and Winands and Nijssen [72] consider the effect of coalitions in the game Scotland Yard. Nijssen and Winands also described a multiplayer version of their MCTS-Solver [73].

3.3 Search in Imperfect Information Games

In this section an overview of existing approaches to performing tree search in imperfect information games is presented.

3.3.1 Determinization

For games of imperfect information, the states of a game are grouped together into information sets. An information set is associated with a particular observer, who cannot distinguish between states in the information set. Since the utility of an action may depend on which state within an information set is the actual state of the game, each player aims to maximize their expected utility where expectation is taken over both states and opponent policies.

One approach to designing AI for games with stochasticity and/or imperfect information is determinization, also known as Perfect Information Monte Carlo (PIMC) [30]. For an instance of a stochastic game with imperfect information, a determinization is an instance of the equivalent deterministic game of perfect information, in which the current state is chosen from the AI agent's current information set, and the outcomes of all future chance events are fixed and known. For example, a determinization of a card game is an instance of the game where all players' cards, and the shuffled deck, are visible to all players, or in a dice game a determinization might fix the sequence of values a die will roll. Determinizations can be samepled from the current game state, and each

one analysed using AI techniques for deterministic games of perfect information. The decisions made in each determinization can then be combined to yield a decision for the original game. The term determinization refers to the process of converting a game of imperfect information to an instance of a game of perfect information. The AI technique of analysing multiple determinizations to make a decision is often called Monte Carlo sampling (of determinizations). In this work Monte Carlo sampling of determinizations is referred to simply as determinization to avoid confusion with the Monte Carlo sampling of game simulations used by MCTS algorithms.

Ginsberg's GIB system [16] applies determinization to create an AI player for the card game Bridge which plays at the level of human experts. GIB begins by sampling a set D of card deals consistent with the current state of the game. For each of these deals $d \in D$ and for each available action a, the perfect information ("double dummy") game is searched to find the score $\mu(a, d)$ resulting from playing action a in determinization d. The search uses a highly optimised exhaustive search of the double dummy Bridge game tree. Finally, GIB chooses the action a for which the sum $\sum_{d \in D} \mu(a, d)$ is maximal.

Bjarnason et al [48] present a variant of UCT for stochastic games, called *Sparse UCT*, and apply it to the single-player card game of Klondike Solitaire. Bjarnason et al [48] also study an ensemble version of Sparse UCT, in which several search trees are constructed independently and their results (the expected rewards of actions at the root) are averaged. They find that ensemble variants of UCT often produce better results in less time than their single-tree counterparts. A special case of ensemble Sparse UCT, which Bjarnason et al call *HOP-UCT*, is equivalent to a straightforward application of determinization (more specifically, hindsight optimization [74]) with UCT as deterministic solver, in which the determinization is constructed lazily as UCT encounters each chance event.

Bjarnason et al [48] treat Klondike Solitaire as a stochastic game of perfect information: rather than being fixed from the start of the game, the values of face down cards are determined as chance events at the moment they are revealed. This works for single-player games where the hidden information does not influence the game until it is revealed, but generally does not work for multiplayer games where the hidden information influences the other players' available and chosen actions from the beginning of the game. It is applicable in some cases, e.g. if players draw cards from a shuffled deck, it can be assumed that the next card is chosen randomly at the moment it is drawn rather than fixing the order of the entire deck in advance. Nevertheless, the specific methods of Sparse UCT and lazy determinization are not immediately applicable to multiplayer games, but the general ideas may be transferable. Bjarnason et al [48] show that Sparse UCT is able to win around 35% of Klondike Solitaire games, which more than doubles the estimated win rate for human players. Determinization is also the state-of-the-art approach for card games such as Bridge [16] and Skat [49, 75]. Determinized MCTS also shows promise in games such as Phantom Go [76] and Phantom Chess (Kriegspiel) [77], as well as the highly complex card game Magic: The Gathering [51, 52].

Despite these successes, determinization is not without its critics. Russell and Norvig [78] describe it (somewhat dismissively) as "averaging over clairvoyance". They point out that determinization will never choose to make an information gathering play (i.e. a play that causes an opponent to reveal some hidden information) nor will it make an information hiding play (i.e. a play that avoids revealing some of the agent's hidden information to an opponent). Ginsberg [16] adds weight to this claim by making the same observations about GIB specifically.

Russell and Norvig's criticisms of determinization are valid but equally valid are the experimental successes of determinization. Frank and Basin [28] identify two key problems with determinization:

- *Strategy fusion*: An AI agent can obviously not make different decisions from different states in the same information set (since, by definition, the agent cannot distinguish such states); however, different decisions can be made in different determinizations.
- *Non-locality*: Some determinizations may be vanishingly unlikely (rendering their solutions irrelevant to the overall decision process) due to the other players' abilities to direct play away from the corresponding states.

Strategy fusion may arise since a deterministic solver may make different decisions in each of the states within an information set. In this situation the issue is that the agent assumes a different decision can be made depending on the state and this information is not known.

Building on the work of Frank and Basin, Long et al [30] identify three parameters of game trees and show that the effectiveness of determinization is related to a game's position in this parameter space. The parameters measure the ability of a player to influence the outcome of a game in its late stages (leaf correlation), the bias in the game towards a particular player (bias) and the rate at which hidden information is revealed (disambiguation). Long et al [30] demonstrate how these parameters can be used to predict whether determinization is an appropriate method for a given game.

3.3.2 Minimax

The minimax algorithm [78](Ch. 6) is commonly used to search game trees in combinatorial games and has been hugely successful such as producing the first program to beat a chess grandmaster [19]. There are several algorithms which extend minimax search to games with stochastic outcomes. Firstly there is expectimax search [78] where the value of a chance node is the expected value of a randomly chosen child (i.e. the sum of the values of its children weighted by the probabilities of the corresponding chance outcomes). Additionally Ballard introduced *-minimax trees [79] for handling chance events. There is also miximax search [80] which is similar to single player expectimax, in which a predefined strategy is used for opponent decisions allowing opponent decision nodes to be treated as chance nodes. Smith and Nau [81] combined Hierarchical

Task Networks (HTN) with Minimax search for the game of Bridge. The HTN was used to restrict the search tree to nodes reached by choosing certain strategies. This is similar to the idea used in the ISMCTS algorithm, of grouping information sets in order to reduce the complexity of the search tree.

3.3.3 Monte Carlo Tree Search

There have been many examples of the MCTS algorithm being combined with PIMC search as a technique for hidden information games. In this thesis, the ISMCTS algorithm is introduced which extends the MCTS algorithm to trees of information sets in order to overcome the problem of strategy fusion in PIMC search. Several other algorithms for applying MCTS to imperfect information games have been developed in addition to the ideas presented in this thesis. The idea of constructing trees of information sets and sampling determinizations to restrict the region to be searched is used by the ISMCTS algorithm. This is similar to the Partially Observable UCT (PO-UCT) approach of Silver and Veness [82] (which also samples from beliefs), although PO-UCT operates on the domain of partially observable Markov decision problems (i.e. 1-player games of imperfect information) rather than adversarial games. Schäfer [75] also applied an information set tree approach for the game Skat using the UCB1 algorithm for selection, where the information sets in the tree are from the point of view of the player about to play. Furtak and Buro [83] introduced imperfect information Monte Carlo search (IIMC) which improves upon PIMC search in the game Skat. Heinrich and Silver have also demonstrated that a variant of the ISMCTS algorithm using one node per information set can be modified to converge to a Nash-Equilibrium in Kuhn Poker [84].

3.3.4 Counterfactual Regret Minimization

One popular approach to AI for imperfect information games is to compute (or approximate) a Nash-equilibrium strategy; examples of this approach include Gala [85] and counterfactual regret (CFR) [36]. Approximating a Nash-equilibrium is desirable in games where exploitability of strategies is an important factor. The CFR algorithm has been hugely successful in the domain of computer poker. Versions of the CFR algorithm which make use of Monte Carlo sampling have also been developed [37], which will converge to a Nash-equilibrium.

To express a Nash equilibrium requires storing a mixed policy for every information set, however in many games the number of information sets is too large for this to be tractable and it is impossible to compute a Nash equilibrium. One technique for dealing with this is to use *abstractions*, which is a smaller game obtained by merging together information sets in a larger game. A policy for the smaller game can be applied to the larger game by using the policy from the smaller game in information sets that are merged together in the abstraction. Abstraction has been successful Poker games [80]. The purpose of this is to reduce the game to a smaller game for which computation methods

are tractable. The drawbacks of this approach are that an abstraction must be designed for each new domain and the Nash-equilibrium computed for the abstraction may not be a Nash-equilibrium in the larger game [86].

Recently Lanctot et al [87] have developed an online sampling approach to conterfactual regret minimiztion which is guaranteed to converge to a Nash-equilibrium. Furthermore this method produces less exploitable strategies than ISMCTS in the game Goofspiel. Approximating a nash-equilibrium will yield a strategy that is robust against being exploited, but will not be able to exploit another policy. This can be addressed with the CFR algorithm if an equilibrium is computed using an opponent model. Shafiei et al [65] demonstrate that CFR can learn to exploit MCTS. In both cases it appears that MCTS produces strategies which are exploitable, however the ideas presented in Chapter 7 (and other techniques such as Smooth UCT [84]) can reduce the exploitability of MCTS. If closely approximating a Nash-equilibrium is important in an application of AI, then counterfactual regret minimization is the best approach. However MCTS scales excellently to large and complex problems and it well suited to learning robust pure strategies, which appear to exist in many popular imperfect information games.

3.3.5 Inference

In games of imperfect information, it is often possible to infer hidden information by observing the moves of the other players, according to some model of the other players' decision processes. One way of capturing this notion is via *belief distributions*, probability distributions over states in the current information set where the probabilities are inferred from the history of observed moves. This type of inference has frequently been applied to the game of poker [88, 89], but also to other games such as Scrabble [90] and the card game Skat [49, 75]. Chapter 7 investigates how inference can be performed by using MCTS as an opponent model.

Chapter 4

Software and Domains

4.1 MctsFramework

In this section an overview of the software framework used to conduct the research in this thesis is presented. The *MctsFramework* was developed in collaboration with Dr. Edward Powley and used in a number of research projects in addition to work in this thesis. *MctsFramework* is written mostly in `C#` with additional components in `C++` and `python`. The purpose of the framework is to facilitate the development of new MCTS based algorithms and be able to perform large scale experiments across a wide variety of domains. The framework accomplishes this by providing an interface between game logic and AI code and a set of tools to automate playing particular games with particular AI players.

The framework uses a model of games based upon states, actions and players which is very close to the model defined in Chapter 2. Adding a new game to the framework requires four main components:

- A function to construct a initial state
- A function to generate legal actions for a given state
- A function to return a new state, given a state and a legal action
- A function to determine is a state is terminal and a function to calculate the rewards for each player

AI players developed in the framework perform one function, choosing an action given a state. AI players that do not have any game specific dependencies (for example the UCT algorithm) can play any game implemented in the framework.

There were many advantages gained from developing *MctsFramework* for the experiments in this thesis. Firstly it enabled a lot of code re-use (particularly of game implementations) which increased the reliability of experiments since most of the code is mature and well tested. Furthermore, for many of the later experiments there was already a large collection of games implemented allowing

more comprehensive experiments to be run. Similarly when implementing a new game there are numerous generic AI players implemented which can be used to test the game logic.

One disadvantage of using the framework is that it adds some performance overhead (which is arguably offset by reduced development time) and makes no attempt to manage memory usage. The framework is ideal for research purposes, but would be unsuitable for direct application in a commercial product. Furthermore there is some crossover between the game logic and AI code in the handling of hidden information, in that games must add intermediate states when information is revealed. The framework has no mechanism for enforcing the sharing or hiding of information from AI players and these intermediate states are used to create non cheating MCTS agents, which is discussed in Chapter 6.

The framework will run a game with a set of AI players and control the flow of the game, passing the state of the game to AI players and applying the moves chosen. Additionally the move history for the game can be recorded and a game can be wound back to an earlier state. When implementing a new game the option exists to supply a text representation of states and actions, which may make use of coloured letters and backgrounds. Similarly an AI player can output text containing information about each decision. When running a game this information is aggregated allowing an observer to follow the progress of a game, or can be turned off to speed up the execution of games. The final major component of the framework is integration with a BOINC [91] server used to run large-scale experiments in a many-CPU cluster.

4.1.1 Components of MctsFramework

MctsFramework contains the following base classes:

- **GameState:** Holds all information about the current state of a game and provides methods to generate the move list, apply moves, determine if the state is terminal and calculate the rewards for each player. Additionally there are methods to get generic information about the state of a game such as the current player and number of players. There are also several utility methods for use with MCTS including getting the result of a random play-out of the game or generating a random move. These utility methods may be overridden for a particular game allowing game specific optimisations to used.

- **GameMove:** Contains information used to specify actions chosen by a player. Also implements equality checks with other GameMove classes.

- **GamePlayer:** Provides an interface to an agent which will return a GameMove given a GameState. Additionally there are methods which instruct AI agents to initialize their internal state and update their internal state in response to actions taken by other players (if necessary).

- **GameManager:** Allows a game to be played out with an initial GameState and a set of GamePlayer implementations. Additionally has the option to output information about the game in progress to the console with varying levels of verbosity.

4.1.2 Handling Hidden Information

In order to handle hidden information, the framework uses GameInfoSet classes to represent information sets. However it would be intractable to store all possible states in an information set, so instead a GameInfoSet stores the actual state of the game, and provides a method to sample determinizations of this state. Additionally a GameInfoSet stores the owner of the information set, so that determinizations can be correctly sampled for a particular observer. The framework also handles partially observable actions, through the use of GameInfoMove classes which contain a GameMove and an owner.

These classes exist to facilitate the implementation of AI players which handle sources of imperfect information, however the framework does not keep any aspect of a state hidden, so AI players can be made to "cheat" if necessary. This also leads to a complication when information is revealed, since the framework does not have an interface for revealing information to AI players. If at any point information is revealed to some players but not others, the GameState must enter a state in which the environment (labelled player 0) chooses a GameInfoMove. Simultaneous moves are also a special case, in which the GameState must indicate that moves are simultaneous, then each player selects a GameMove sequentially (which can be a null action if a player is not required to make a choice) each of which is applied as a regular action, before entering a state where the environment (player 0) plays a GameMove which corresponds to the set of choices by each player. Chance events are also implemented by having the environment (player 0) chose an action. In all experiments the AI used for player 0 is a GamePlayer which chooses randomly amongst all legal actions.

4.1.3 Card Games Framework

The Card Game Framework is a sub framework which provides a base for implementing games which are played using a standard French deck of 52 cards. A card game is represented as a finite set of collections of cards, each of which has a list of players who can observe the contents. There are also container classes for ordered, unordered and singleton sets of cards. Queries on sets of cards can be made with these containers which do not depend on the underlying representation. This greatly simplifies the implementation of game logic, whilst taking advantage of the fact that operations on unordered sets of cards can be performed extremely quickly.

The card game framework also provides an algorithm for generating determinizations of a card game state which is based upon rejection sampling. This algorithm can take into consideration hard constraints on which cards are

allowed in which sets and which cards are visible to particular players. Additionally the card game framework provides a method for dealing cards, which will ensure that a determinization will fix all future card deals.

4.1.4 Integration with BOINC

The majority of experiments in this thesis consist of a set of AI players playing a number of games with various parameters. In order to run such experiments, the individual games were created as a BOINC work units. Each work unit contains the framework code and a JSON file from which the game and the AI players with particular parameters can be constructed. The JSON file also specifies where the result will be stored in a results database. A python script can be written for each experiment, which creates the JSON files submits them as work units on the BOINC server. Finally another python script can be written to query results from the results database, perform any required data processing and generate graphs. This process for automating experiments was continuously developed and improved whilst conducting the research presented in this thesis. In many cases experiments requiring CPU-years were run over a weekend.

This chapter presents an overview of the software framework developed for running the experiments presented in this thesis. Additionally an overview of the rules for each of the domains tested is given, along with any important implementation details.

4.2 Domains

This section describes each of the domains used for experimentation in this thesis including an overview of the rules and any important implementation details. In addition a summary of other work on developing AI for these domains is provided where relevant.

4.2.1 Dou Di Zhu

Dou Di Zhu is a 3-player gambling card game which originated in China, which falls into the class of ladder games (where players must play a "higher" ranked set of cards on their turn or pass). The name Dou Di Zhu translates into English as "Fight The Landlord" and is a reference to the class struggle during the Cultural Revolution in China where peasants were authorized to violate the human rights of their Landlords. In the original version of the game, studied in this thesis, two players compete together against a third player, the *Landlord*. There are other versions of the game involving four and five players but these are less popular.

The game was only played in a few regions of China until quite recently, when versions of the game on the internet have led to an increase in the popularity of the game throughout the whole country. Today Dou Di Zhu is played by

millions of people online, although almost exclusively in China, with one website reporting 1 450 000 players per hour. In addition there have been several major Dou Di Zhu tournaments including one in 2008 which attracted 200 000 players.

Dou Di Zhu is interesting from an AI perspective as it necessitates both competition (between the Landlord and the other two players) and cooperation (between the two non-Landlord players).

Rules

Dou Di Zhu uses a standard 52 card deck with the addition of a black joker and a red joker. A brief description of the rules is given here; a complete description can be found in [92]. Suit is irrelevant but the cards are ranked in ascending order $3, 4, \ldots, T, J, Q, K, A, 2$. A bidding phase (which is not considered) designates one of the players as the *Landlord*. The Landlord receives 20 cards dealt from a shuffled deck, while the other players receive 17 each. The goal of the game is to be the first to get rid of all cards in hand. If the Landlord wins, the other two players must each pay the stake to the Landlord. However if either of the other two players wins, the Landlord pays the stake to both opponents. This means the two non-Landlord players must cooperate to beat the Landlord. The non-Landlord players do not see each other's cards, so the game cannot be reduced to a two-player game.

Card play takes place in a number of rounds until one player has no cards left. The Landlord begins the game by making a *leading play*, which can be any group of cards from their hand provided this group is a member of one of the legal move categories (see Table 4.1). The next player can play a group of cards from their hand provided this group is in the same category and has a higher rank than the group played by the previous player, or may pass. A player who holds no compatible group has no choice but to pass. This continues until two players pass, at which point the next player may start a new round by making a new leading play of any category.

One exception to the rule that successive plays are of the same type is that a Bomb or a Nuke may be played at any point. Only a Bomb of higher rank or a Nuke can follow a Bomb, and no move can follow a Nuke. Some categories allow extra kicker cards to be played with the group which have no effect on the rank of the move being played. If a move with kickers is played, the next player must play a move in the same category with the same number of kickers.

Making a leading play is a good position to be in, allowing a player to choose a move type where he holds multiple groups, or holds a high-ranking group that opponents are unlikely to be able to follow. The two non-Landlord players also need to work together since they either both win or both lose.

Implementation

The bidding phase is omitted from the implementation, instead assigning an arbitrary player as the Landlord. This allows algorithms to be compared based the strength of on card play alone. Also determinization is carried out in the

Name	Description
Solo	Any individual card, for example A or 2. It is also possible to play runs of sequential cards with length at least 5, for example 345678 or 89TJQKA.
Pair	Any pair of identically ranked cards for example 55 or 77. It is possible to play runs of sequential pairs with length at least 3, for example 334455 or TTJJQQKK.
Trio	Any three identically ranked cards for example AAA or 888. It is possible to play runs of sequential trios of any length, for example 444555 or TTTJJJQQQ. Each trio may also be played with a single *kicker* (single card or pair). In a sequence of trios kickers are either all pairs or all single cards, with all (or none) of the trios having kickers. For example 444555TJ (two single card kickers) or 999QQ (one pair as a kicker).
Quadplex	Any four identically ranked cards with two kickers of differing rank attached, for example 4444TJ or 999955KK.
Bomb	Any four identically ranked cards, for example 5555 or 2222.
Nuke	The red joker and the black joker together.

Table 4.1: Dou Di Zhu Move Categories

natural way, with all hidden cards from the point of view of a particular player being randomly reassigned amongst opponents.

The branching factor for leading plays is typically around 40, and for non-leading plays is much smaller. However, in situations where moves with kickers are available each combination of move and kicker must be considered as a separate move, leading to a combinatorial explosion in the branching factor for leading plays. It should be noted that this is a problem specific to Dou Di Zhu caused by the game mechanic of being able to attach kicker cards to a play. To ameliorate this, an approach similar to the move grouping approach of Childs et al [93] is used: the player first chooses the base move and then the kicker, as two separate consecutive decision nodes in the tree. This adds an extra layer of nodes to the tree (one for each base move), but reduces the branching factor at each node.

4.2.2 Mini Dou Di Zhu

In one experiment a simplified version of Dou Di Zhu is used that removes some of the complexity of the full game and is small enough to be solved with exhaustive tree search techniques, while still retaining some of the strategic depth of the full game. Mini Dou Di Zhu is a 2-player game, played with a reduced deck of 18 cards (four ranks and two jokers). Each player receives seven cards, and the four remaining cards are hidden from both players. There are four move categories, consisting of 1, 2, 3 or 4 cards of the same rank. As

with the full game, the aim is to be the first player to play all cards, but unlike the full game there is no element of cooperation. The total number of distinct deals in Mini Dou Di Zhu is 8832. The game trees for the perfect information variant are small enough that minimax search can be used to exactly determine the game theoretic value of each perfect information deal; when the non-uniform probabilities of obtaining each deal by shuffling and dealing cards are taken into account, approximately 70.7% of games are wins for player 1.

4.2.3 Hearts

Hearts is a four-player trick taking card game played with a standard 52-card deck made popular through a version distributed with Microsoft Windows, where players accumulate points and the goal of the game is to minimize the number of points taken. Cards in the ♡ suit have a point value of 1 each and the $Q\spadesuit$ card has a point value of 13. The goal is to score as few points as possible, i.e. to avoid winning tricks with those cards in them. There is one exception to this goal: taking all thirteen ♡ cards and $Q\spadesuit$ in a round is called *shooting the moon*, and causes every player to score 26 points except the player who shot the moon. Shooting the moon is a risky strategy: the reward for success is high, but so is the cost of failure. Another rules in Hearts is that at the start of each round, players pass 3 cards from their hand to another player which shifts each round (so that every fourth round no cards are passed). The passing of cards introduces partially observable moves into Hearts. For a complete description of the rules see [94].

Previous work on MCTS for Hearts [95, 69] has treated the game as one of perfect information, i.e. played with all cards face up. In this thesis ISMCTS is develop to handle the imperfect information explicitly.

Hearts is played over several rounds, the game ending when any player reaches a score threshold (50 points in experiments). When simulating random games of Hearts, it would be possible to simulate only to the end of the current round and have players seek to minimise their per-round score. However this removes some of the strategic richness from the game, as certain decisions (such as whether to force $Q\spadesuit$ upon a certain player, or whether to attempt shooting the moon) can depend on the overall game score. To capture this, a large number of rounds (10 000 for experiments) was simulated offline, and construct a database of per-round scores. When performing simulated games online, simulations run to the end of the current round as usual, and then sample round scores from this database are used until the score threshold is reached. This is equivalent to simulating to the end of the game, but much more efficient. At the end of the game a score of 1 is given for being first, 0 for being last and $\frac{1}{2}$ for placing second or third. This ensures that if winning is not possible, AI agents are incentivized to not place last.

The tree structure of Hearts is very regular, since during card play there are always 13 cards played by each player (so a depth of 52 moves) and the branching factor can never exceed 13, although most of the time the number of legal moves is smaller since players must follow suit. When deciding which

cards to pass at the start of the round, rather than branching for all possible selections of 3 cards from 13, players select the individual cards sequentially (similar to move grouping [93]). The playing strength of ISMCTS (introduced in Chapter 5) for Hearts it not significantly different to the AI in the Microsoft version, winning approximately 25% of games.

4.2.4 Spades

Spades is a 4-player trick taking card game which originated in the United States in the 1930s but has since spread worldwide [96]. Spades shares many similarities with the game of Bridge, with equally deep strategy yet slightly simpler rules. The players are named after the compass points, with North and South forming a coalition against East and West. A game is played across multiple *rounds* where each partnership receives a score at the end of each round. The winning partnership has the highest score when one or both partnerships exceed 500 total points at the end of a round.

At the start of a round each player is dealt a 13 card hand from a standard 52 card deck. In turn, players provide a single *bid* which is an estimate of how many *tricks* they expect to take from their hand that round. Each trick consists of each player in turn playing a card out onto the table. One of the players (rotating between rounds) is designated the *leader* and may play any card (with the exception that ♠ cards cannot be led until *broken*, i.e. until a ♠ is played as a non-leading card). Subsequent players must match the suit of that card if they can. If a player cannot follow suit, they may play any card. The winning card is the one with the highest rank matching the suit played by the leader, unless a ♠ card was played in which case the ♠ *trumps* the other suit (and the highest ranked ♠ wins the trick instead). The winner of the trick becomes the leader of the next trick. Once a ♠ card has been played (referred to as *breaking* spades), players can lead with ♠ cards for the remainder of the round. The round ends when all players have played all their cards, so a round consists of 13 tricks.

After all thirteen tricks have been played out, scores are calculated as follows for a partnership with a total bid of b, which won a total of t tricks:

- If $t \geq b$ then the partnership earns $10b$ points
- If $t < b$ then the partnership loses $10b$ points
- If $t > b$ then the partnership receives $t - b$ *bags*
- If they have 10 or more bags, the partnerships loses 10 bags and 100 points
- If a player in the partnership bid 0 ("nil") and personally took 0 tricks, the partnership receives 100 points
- If a player in the partnership bid 0 and personally took at least 1 trick, the partnership loses 100 points

Simulated random games in Spades need only run up to end of the current round. Simulating beyond the end of the current round is unnecessary since future deals are unknown, however the current distribution of points may influence the strategy used. Simulated games are therefore terminated at the end of the current round and the following value awarded to each partnership:

$$\frac{(s - 10b) - (s_o - 10b_o)}{200} \tag{4.1}$$

where s is the partnerships score and b is the number of bags the partnership has taken (and s_o, b_o are the score and number of bags for the opposing partnership respectively). The value 200 scales the evaluation for each player to an interval of size approximately 1, which avoids the need to retune the UCB1 constant when using MCTS. It is technically possible for the score difference to change by more than 200 in a single round, but very unlikely. Since gaining 10 bags incurs a penalty of 100 points, the value includes a penalty of $\frac{100}{10} = 10$ points for each bag. This value captures several important tactical ideas in Spades, for example attempting to make your own bid and preventing your opponents from making theirs.

The ISMCTS algorithm (introduced in Chapter 5) was tested against a commercial AI opponent developed by AI Factory Ltd for their popular Spades product on Android mobile devices. The AI Factory AI opponent uses flat Monte Carlo evaluation with a hand-crafted heuristic simulation policy. ISMCTS significantly outperformed the commercial AI, winning $58.3\% \pm 3.1\%$ (95% confidence) of games. ISMCTS is stronger than the AI Factory player at card play, but weaker at bidding: a hybrid player using ISMCTS for card play and AI Factory's heuristic approach for bidding wins $69.1\% \pm 2.9\%$ of games against the AI Factory player. Despite this, experiments in this thesis use ISMCTS both for bidding and for card play. A version of ISMCTS enhanced with heuristic knowledge is deployed in the current release of AI Factory Spades [10]. The heuristic knowledge is primarily used to alter the playing style for a more enjoyable game; in terms of pure playing strength, ISMCTS provides strong performance without knowledge.

4.2.5 Lord of the Rings: The Confrontation

Lord of the Rings: The Confrontation (LOTR:C) [64] is a two-player strategy board game themed on J. R. R. Tolkien's "The Lord of the Rings" novels.. Each player has nine character pieces, each with its own strength value and special ability. Each player can see the identities and locations of his own pieces, but only the locations of his opponent's pieces. If a player moves a piece into a square occupied by his opponent, combat ensues: the identities of the attacking and defending pieces are revealed, and the players simultaneously choose a card each which affects the outcome of combat. The two players have asymmetric win conditions: the "Light" player wins by moving one of his pieces (Frodo) into the opponent's home square (Mordor), whereas the "Dark" player wins by killing Frodo in combat.

4.2.6 Rules

The game-play of LOTR:C has common features with Stratego [97], where identities (but not locations) of a player's pieces are hidden from the opponent. Furthermore, the identity of a piece specifies certain unique characteristics. LOTR:C is an interesting game from an AI point of view since it features hidden information, chance events, partially observable moves and simultaneous moves. It is also asymmetric since both players have different win conditions and thus require different tactics and strategies.

Game Structure

The game is played on a 4×4 grid, with the players' home squares at opposite corners. Most squares can be occupied by more than one piece simultaneously, subject to restrictions. The players are designated Light and Dark, with Dark playing first. Each player has nine character pieces, which they place on the board at the start of the game subject to certain constraints. Each character has an associated strength value between 0 and 9, and a special ability that changes the rules of the game in certain situations. Light's characters are different from Dark's. Generally characters move one space at a time towards the opponent's home square, although some characters and some squares on the board allow for different moves.

The identities of a player's characters are hidden from the opponent until revealed in combat. This leads to a source of hidden information, where the information set specifies the number of opponent pieces in each square and the states in the information set specify the identity of all the pieces. When an opponent moves one of their pieces, this move is partially observable since a player knows a piece moved (and this leads to a new information set) but only the opponent knows which piece moved. Knowledge about the locations of opposing characters can decrease as well as increase. For example if a character whose identity is known enters a square with an unknown character then later exits the square, the identities of both the exiting character and the remaining character are unknown. Since players must move pieces forwards (aside from a few special rules), the LOTR:C game tree has very few cycles and random games are almost always fairly short.

Objectives

LOTR:C has multiple win conditions, which differ for each player. For the Light player there are three ways to win:

- Moving the character Frodo into Dark's home square;
- Killing all Dark characters;
- The Dark player being unable to move any characters.

For the Dark player there are also three ways to win:

- Killing the character Frodo;

- Moving any four characters into Light's home square;
- The Light player being unable to move any characters.

Combat

When a character moves into a square that contains opponent characters, combat is initiated. The moving character becomes the attacker and a randomly chosen opponent character in the square is the defender, then both players simultaneously choose one of the combat cards from their hand. This leads to simultaneous moves being a feature of the game. Each player begins with nine cards (which are removed once played) and each character has a strength value, as do some of the cards. In combat the player whose combined character and card strength is greatest wins the combat. Some characters and some cards feature text that can alter the outcome of the combat, by either offering a player extra choices or altering the rules of combat. Typically the outcome of combat is that one or both characters is defeated and removed from play.

Implementation

Character movement in LOTR:C is partially observable. Therefore actions are defined such that they identify the character and the source and destination squares (e.g. "move Frodo from Cardolan to Eregion"). The move observed by the opponent does not identify the character (e.g. "move a character from Cardolan to Eregion").

Some care is needed to ensure the structure of the game tree, particularly around combat, conforms to that described in Section 2.2. An environment player is used to model actions taken by the game. Specifically the environment player is responsible for deciding the outcome of chance events and for revealing information to players. In the implementation, a typical instance of combat consists of the following sequence of actions:

1. The attacking player moves a piece into a square occupied by an opponent piece.
2. The environment player reveals the identities of the attacker and defender pieces, choosing a defender at random if necessary (which leads to a source of chance events).

3, 4. Both players simultaneously choose a card.

5. The environment player reveals the chosen cards and resolves the combat.

A skilled human player of LOTR:C remembers the information revealed about the identities of characters. The implementation enforces perfect recall for all players: information about which characters can possibly occupy which squares based on previously revealed information is encoded in the game state. In particular generated determinizations are always consistent with this information.

Initial setup

Before each game, players can place their characters on the board in any configuration subject to certain constraints. The choice of initial setup has important strategic consequences, however tree search is not well-suited to solving this problem: each player has a choice between $\frac{9!}{4!} = 15\,120$ possible initial setups for their pieces, and both players choose simultaneously. Evaluating the effectiveness of each configuration would require a large amount of computational effort and then more computational effort would need to be spent finding a good policy for selecting a configuration. This problem is not tackled, instead all experiments were conducted on a single, hand-designed initial setup intended to be typical of those that a pair of human players might choose. This re-use of the same initial setup also has the effect of reducing the variance in experimental results. No information persists between trials, so there is no danger of the algorithms adapting themselves to this particular setup.

4.2.7 Phantom m, n, k-games

An *m, n, k-game* [98, 99] is a two-player game played on an $m \times n$ grid. Players take alternating turns to mark a square. The winner is the first player to mark k squares in a horizontal, vertical or diagonal row. For example, the well-known game of Noughts and Crosses (or Tic-Tac-Toe) is the $3, 3, 3$-game, and Go-Moku [100] is the $19, 19, 5$-game.

A *phantom m, n, k-game* is an m, n, k-game in which neither player can see the positions of the opponent's marks. If a player tries to mark a square that is already occupied by his opponent, the player is told that this is an invalid action and is allowed to choose again. There is no penalty associated with playing an invalid move. Indeed, playing invalid moves is the only mechanism by which the phantom m, n, k-game player can gain information about his opponent's previous plays, so doing so is never detrimental and often beneficial. In terms of the game tree, each player action is followed by an environment action specifying whether the move is valid or invalid.

There appears not to be any previous study of phantom m, n, k-games in the context of MCTS, although phantom Tic-Tac-Toe (i.e. the phantom $3, 3, 3$-game) has been studied by Auger [101] and by Teytaud and Teytaud [102], and other phantom games have been studied by Borsboom et al [76] as well as Ciancarini and Favini [103, 77]. In this thesis experiments are perfomed on the phantom $4, 4, 4$-game (which has just enough states that the algorithms do not exhaustively search the full perfect information tree given the iterations budget).

The perfect information $4, 4, 4$-game is known to be a draw [98]. However this analysis does not carry over to the phantom version of the game: intuitively, even a perfect (but non-cheating) player cannot block a line they cannot see. There does not appear to be a theoretical analysis of the phantom $4, 4, 4$-game; but it appears based on empirical evidence is that the game has no forced result, and while player 1 has a strategy that can lead to a fast win (create four in a

row as quickly as possible, hoping that player 2 does not discover or block the line) the game is somewhat balanced overall.

4.2.8 Checkers

Checkers (or English Draughts) is a two-player game of perfect information, played on an 8×8 board with 12 pieces per player. Pieces may be moved forwards to a diagonally adjacent empty square, or may jump diagonally forwards by two squares if the target square is empty and the intervening square contains an opponent piece. Jumping over an opponent's piece causes it to be *captured*, and removed from the game. Captures may be chained together, if the jumping piece can immediately capture another piece. Otherwise the turns alternate between the two players after each move. In the variant of Checkers studied in this thesis, captures are forced: if a capture move is available then it must be played, although if more than one is available the player may choose which one to take. If a piece moves onto the opponent's home row, it becomes *crowned* and may subsequently move and capture backwards as well as forwards. A player wins by leaving their opponent with no legal moves, i.e. by blocking or capturing all their pieces.

Draws (stalemates) are common in checkers; indeed, perfect play by both sides will always lead to a draw [20]. AI programs capable of perfect play exist, such as Chinook [20]. As Checkers was solved more than a decade before the invention of MCTS, there has been little work on developing strong MCTS players. However Checkers is often used as a test domain for enhancements in General Game Playing systems [104, 63].

4.2.9 Othello

Othello (or Reversi) is a two player game with perfect information, played on an 8×8 board. The game starts with the centre four squares of the board containing two black and two white pieces placed diagonally opposite each other. A move consists of placing a piece on the board; for the move to be legal, it must sandwich a horizontal, vertical or diagonal line of one or more opponent pieces between the newly placed piece and an existing own piece. The sandwiched pieces are captured, and converted to the colour of the player who moved. If (and only if) a player has no legal moves, he must pass; when both players pass consecutively, the game is over. The player with the most pieces on the board wins the game.

Strong Othello programs exist which are capable of beating the strongest human players, one of the first such programs being Logistello [105]. More recently, MCTS has been combined with offline learning methods to produce strong play [106, 107, 108].

4.2.10 Backgammon

Backgammon is a two player game which has stochasticity in the form of dice rolls, but otherwise has perfect information (i.e. there is no information hidden from one player but visible to another). The board has 24 spaces, which are numbered 1–24 in opposite directions for the two players. Each player begins with 15 pieces in a standard initial setup. The aim of the game is to move all of one's pieces towards space 1 and off the end of the board. A player's turn begins with a roll of two dice. The player then takes two moves, one for each of the two rolled numbers, moving a piece forward the given number of spaces. The same piece can be moved twice in one turn. If the two dice have the same number, the player makes four moves instead of two.

A piece cannot be moved to a space occupied by two or more opponent pieces. However a piece can be moved to a space occupied by a single opponent piece, in which case the opponent piece is captured and moved to the *bar*, equivalent to space number 25. If a player has pieces on the bar, they must move them back onto the board before they may move any other pieces. A common basic strategy in Backgammon is to force the opponent to skip several turns, by capturing a piece having blocked the spaces into which it could be moved back onto the board. When all of a player's pieces are on spaces 1–6, pieces may be moved off the board (beyond point 1) and removed from the game. The first player to remove all their pieces in this way is the winner.

Strong AI players for Backgammon, such as TD-Gammon [109], are capable of beating the strongest human players. MCTS has also been demonstrated to produce strong decisions in Backgammon [110].

4.2.11 The Resistance

The Resistance [111] is a game for 5 to 10 players where players play one of two roles, spies and non-spies. The non-spies aim to successfully complete 3 out of 5 missions whereas the spies aim to sabotage 3 out of 5 missions. Each mission is carried out by a team consisting of a subset of players in the game which is chosen by a designated leader and then voted on by the rest of the players. The key mechanic of the game is that the spies are aware of each others' identity but the non-spies are not. Bluffing and inference is therefore a key aspect of the game, since non-spies need to infer the spies' identities (and choose teams without spies) and spies must bluff their way onto missions to carry out sabotage. The game has hidden information, partially observable actions and simultaneous actions.

The turn structure for The Resistance is shown in Algorithm 1. The non-control-flow statements in this pseudocode correspond one-one with the plies of the game tree. Team choice is treated as a single move: the decision node has a branch for every possible team. Voting and mission card choice moves are hidden from other players; all other moves are fully observable. The game implementation keeps track of hard constraints by storing the set of possible spy configurations (from the point of view of an outside observer): when a mission

Algorithm 1 Turn structure for The Resistance.

```
1: while neither team has won three missions do
2:     repeat
3:         the leader chooses a team
4:         if this is not the fifth attempt at choosing a team then
5:             for each player simultaneously do
6:                 the player chooses a vote card
7:             the vote cards for all players are revealed
8:     until the vote passes or this is the fifth attempt
9:     for each player on the team simultaneously do
10:        the player chooses a mission card
11:    the number of sabotage cards are revealed
```

ends with $k > 0$ sabotages, all configurations with fewer than k spies on the mission team are removed from the set.

The Resistance is a good game for testing how well MCTS works as an opponent model for inference. Firstly because good performance can be achieved with a feasible number of MCTS iterations. This is due to two factors, firstly the small number of information sets (6 for the 5 player game with a small number of states per information set) and both a low branching factor and shallow tree. Secondly bluffing and inference are crucially important to good play in The Resistance. Experiments on the resistance (Section 7.5.1) show MCTS players which perform no bluffing or inference are easily exploitable, which matches the recent results of Lanctot et al [87].

4.2.12 Scotland Yard

Scotland Yard [112] is a board game first published in 1983 (and winner of the "*Spiel des Jahres*" award) where a team of players perform the role of detectives trying to catch another player — a criminal known as Mr X. The game takes place on a graph which represents the streets of London and is similar to the class of hide and seek games on graphs. The detectives are referred to as seekers. Edges on the graph are labelled with modes of transport (taxi, bus, underground and boat) which must be paid for using tickets. Each player begins the game with a number of each ticket type. The seekers do not know the location of Mr X except on turn numbers 3, 8, 13 and 18 when Mr X "surfaces" (reveals his location). The seekers know which type of ticket Mr X used each turn. The game ends either when the seekers are unable to move given their remaining tickets (in which case Mr X wins) or when a detective enters the location containing Mr X (in which case the seekers win).

In Scotland Yard there is an information asymmetry since Mr X knows the location of the seekers. The seekers can form a constraint set on the possible locations of Mr X by observing the modes of transport used given the last known location of Mr X. It has been shown that constructing this constraint set

can be used to improve the performance of MCTS at Scotland Yard [72], and the implementation only samples Mr X locations from this constraint set when generating determinizations. The size of the information set changes throughout the game and may contain between 1 and 199 states (depending on the set of possible Mr X locations). The constraint set gives an opportunity for inference and bluffing which is examined in Chapter 7, since it may be possible to infer that some locations are more likely than others. Scotland Yard also provides an opportunity to investigate how well MCTS scales as an opponent model in games with larger numbers of states per information set.

Moves in Scotland Yard specify the destination node and the ticket type. If a move is partially observable, the destination node is hidden but the ticket type is visible. Seeker moves are always fully observable. The turn structure is straightforward, with no environment moves: at times when Mr X surfaces, his move is simply treated as fully observable. Scotland Yard can be thought of as a two-player game, where one player controls Mr X and the other controls all the seekers. When humans play Scotland Yard it is common for the seekers to be a team, with each member controlling one seeker. This adds an enjoyable social aspect to the game, but is not needed for AI-versus-AI play: the seekers all have the same information and the same rewards, so a team of rational players is equivalent to a single rational player. In all experiments the same AI technique is used for all seeker decisions, thus treating the seeker team as a single player. An alternative approach is called *coalition reduction* [113, 72]: the seeker who captures Mr X is given a slightly higher reward than the others. Suitably tuned, this increases the seekers' win rate.

4.2.13 Saboteur

Saboteur [114] is a hidden identity card game with some similarities to The Resistance in that players belong to teams: Saboteurs and Diggers. The Diggers aim to dig a tunnel to a gold vein whilst the Saboteurs aim to prevent the Diggers reaching gold. Unlike The Resistance however, the members of the subverting team (the Saboteurs) are unaware of each others' identity. The game is played with cards, with each player holding a hand of six cards which are drawn from a common stock. There are also three goal cards (only one of which contains gold) placed in random locations and a ladder card representing the entrance to the mine. The diggers hope to connect their ladder entrance to the gold. Some cards enable players to place tunnels on the table. Other cards allow players to remove sections of tunnel, prevent other players from placing tunnel cards, or secretly look at the goal cards.

In Saboteur there are numerous sources of hidden information and uncertainty. The identities of players are hidden and not revealed until the end of the game, the cards are drawn from a shuffled deck, the location of the gold is randomly determined at the start of the game and any cards discarded by players are hidden. This creates a combinatorial explosion in the number of states per information set.

Bluffing and inference are key aspects of the game, since Saboteurs can be

prevented from influencing the game if they are too obvious in their attempts to sabotage the game. Therefore good saboteurs will be careful not to give away their identity. Despite the similarities, the game tree in Saboteur is significantly more complex than The Resistance, with high branching factors at opponent decision nodes due the large number of different cards they could hold and the large number of ways each card can be used. Saboteur is included as an example to test the scaling of MCTS as an opponent model for inference when the tree search model may be weak due the complexity of the tree being searched and as a game where players must actively bluff in order to be sucessful.

4.2.14 Other Implemented games

The following is a list of games currently implemented in the software framework which were not used for experiments presented in this thesis.

- Arimaa (`http://boardgamegeek.com/boardgame/4616/arimaa`)
- Blob (`http://boardgamegeek.com/boardgame/1116/oh-hell`)
- Connect4 (`http://boardgamegeek.com/boardgame/2719/connect-four`)
- Goofpsiel (`http://boardgamegeek.com/boardgame/16632/gops`)
- Kuhn Poker (`http://en.wikipedia.org/wiki/Kuhn_poker`)
- Liar's Coins (simplified version of Liar's Dice) (`http://boardgamegeek.com/boardgame/45/liars-dice`)
- Nannon (`http://boardgamegeek.com/boardgame/21346/nannon`)
- Nim (`http://boardgamegeek.com/boardgame/11753/nim`)
- Rock, Paper, Scissors (`http://boardgamegeek.com/boardgame/11863/rock-paper-scissors`)
- Urban Rivals (simplified version) (`http://www.urban-rivals.com/`)
- Stratego (`http://boardgamegeek.com/boardgame/1917/stratego`)

Chapter 5

Hidden Information and Strategy Fusion

This chapter describes work investigating how the MCTS algorithm can be used for decision making in games with hidden information. Firstly work investigating the combination of MCTS with perfect information monte-carlo search (PIMC) in the card game Dou Di Zhu [50] is presented. Then a new algorithm, information set MCTS (ISMCTS) [26, 27] is introduced which addresses some of the shortcomings of PIMC search by searching trees of information sets. Recall from Section 2.1 that hidden information is some aspect of the state of a game which is unobservable from the point of view of at least one player. An information set of a player is a collection of states, where the player knows the game is in one of the states within the information set, but not which state. A useful example of a game with hidden information (which is used throughout this chapter), is a card game in which each player has been dealt a hand of cards. The information set belonging to a player contains a state corresponding to each possible distribution of unseen cards amongst the other players hands.

A card deal can be thought of as an action performed randomly by the environment, selecting amongst all possible card deals. If the environment is considered to be an extra player that behaves randomly as defined in Chapter 2.2, then in a hidden information game all information is hidden or revealed when players take actions. This is because actions encode a set of possible state transitions from an information set, leading to a new information set. Revealing information is sometimes described as disambiguation [30] and corresponds to a reduction in the size of an information set. An example might be a player revealing 8♠ from their hand. The other players may then conclude that they are in a state in which that player was dealt 8♠ and can disregard any others. Conversely information can be hidden, which corresponds to an increase in the size of an information set. For example if the player who revealed 8♠ shuffles a card from their hand into a deck and draws a new card, the other players no longer know whether the hand contains 8♠.

The dealing of cards can be thought of as a chance event which occurs at the start of the game. If tree search is to be applied to such a game, including the state before cards are dealt in the tree presents a problem. In a card game using a standard French deck there are $52! \approx 8 \times 10^{67}$ possible orderings (and deals in the case the order of cards dealt matters), before even considering all of the possible states for each deal. For comparison, this is larger than the estimated number of states in Chess (10^{47} [13]) and the state complexity of the largest game ever solved, Checkers (10^{20} [20]). Therefore even enumerating all possible branches at the first node is intractable without performing any analysis of the decision making for each deal. Even if a player cheats and knows the exact state of the game, in some domains the resulting perfect information game may still be highly complex and difficult to solve (although in some domains e.g. Bridge the perfect information games can be solved [16]). At first it appears that analysing every state is unnecessary since many will be impossible as the number of possible states is reduced for a player once their hand has been dealt. However in order to accurately model the decision making of the other players, it must be considered that their information sets could contain potentially any legal state, even those which are not in the decision making players information set. As a consequence, exact solution methods for hidden information games are currently limited to games small enough to enumerate.

In practice it is often the case for many games with hidden information, that decisions must be made without an analysis of every possible state. However it is often the case that learning about one state can tell us something about another state. (The transfer of learning across states is discussed in depth in Chapter 8.2). For example in a card game if there is a card which will win the game, that is often a good card to play in any state where the card is in a player's hand. This property of games can be exploited in order to make decisions without considering every possibility.

A popular technique for hidden information games is *Perfect Information Monte-Carlo search* (PIMC), where sampled determinizations are analysed to produce a decision for each determinization using any standard technique for perfect information games. Then a decision is made based upon the decisions made for each determinization, for example by taking the action most frequently chosen across all sampled determinizations. Determinizations and PIMC search are introduced in Chapter 3.3.1. This approach has been hugely successful for domains such as Bridge [29, 16], Skat [49, 75], Solitaire [48], and Probabilistic Planning [115]. However this method has many drawbacks and has been described as "Averaging over clairvoyance" and is incapable of bluffing [78]. In addition PIMC suffers from the problems of strategy fusion and non-locality [28]. These problems are described in more detail in Chapter 3.3.1. This thesis aims to address each of these issues with MCTS based algorithms.

In this Chapter first the combination of PIMC with MCTS is investigated in Section 5.1. Early work regarding the trade off between simulations and determinizations is presented in Section 5.1.1. Next experiments to quantify the effect of strategy fusion and non-locality in Dou Di Zhu are presented in Section 5.1.2. Section 5.2 introduces one of the main contributions of this thesis,

the ISMCTS algorithm. Experiments from comparing performance of ISMCTS in comparison to PIMC are presented in Section 5.2.4. Finally a summary of the results is presented in Section 5.3.

5.1 Perfect Information Monte Carlo Search

Perfect Information Monte Carlo search (PIMC) is a technique for decision making in games with hidden information. As described in Section 3.3.1, PIMC samples determinizations of the current state of the game, produces a decision for each perfect information game, then combines the decisions for each determinization into a final decision. This approach has been successfully applied to a variety of games, producing world champion Bridge and Scrabble programs, vastly outperforming human players in Solitaire games and has been successful in Probabilistic Planning competitions.

However PIMC is not without flaws. Russel and Norvig describe the approach as "Averaging over Clairvoyance" [78] and Frank and Basin point out two major issues, namely strategy fusion and non-locality [28]. PIMC is not capable of bluffing or making any information gathering or hiding plays, since each determinization is a perfect information game there is no information to gather or hide and it is assumed that players known each others hidden information. All of these issues stem from the fact that PIMC is based on the assumptions that actions which are good in particular determinizations will be good moves in the hidden information game from which the determinizations are derived, yet there are many types of situation in which this assumption is false.

Strategy fusion is the assumption that the player can choose different actions depending on what state the game is in. In a determinization, there is only one possible state but in the hidden information game there are many possible states. When the utility of an action is not consistent across states, PIMC assumes that the action can be selected when its utility is good and avoided when its utility is bad. For example consider a game where a coin is flipped but kept hidden and the player is offered the choice between a payment of 0.9 or a payment of 1 if the player can correctly guess the face up side of the coin and 0 otherwise. The payment of 0.9 is the best, since the player has has an expected reward of 0.5 when guessing the face up side of the coin. However if PIMC is applied to this game, in each determinization it is known which way up the coin is and the guess the coin option always has a reward of 1.

Non-locality is an issue that arises since history can matter in a hidden information game. For example, if a player could have played a winning card on their previous turn and is playing rationally, the player does not hold the winning card otherwise the game would be over. However PIMC would include determinizations in which that player held that card, which is in fact incompatible with the assumption of *rationality* (which asserts that players do not choose actions that offer strictly lower utility than other actions). In a perfect information game the best strategy does not depend on history of previous actions that led to a state, but in a hidden information game it can. Note that non-locality

arises from the assumption that players are rational, which is subtly different to the technique of inference where an opponent model is exploited to determine which states are likely.

Finally since PIMC analyses perfect information games, there is never any information to hide or gather. Therefore PIMC is blind to the utility of bluffing or information gathering actions. In addition it is assumed the opponents cheat since the only determinizations sampled are from the player's own information set. Experiments in this thesis show that this assumption is not a large problem in games where knowledge of hidden information is not very important, in particular the results in Section 5.1.2 indicate that this is the case for Dou Di Zhu. However in other types of games such as poker and hidden role games, the game is degenerate if players are assumed to cheat. For example in the game The Resistance, if the non-spies are assumed to know the identity of the spies, the spies would conclude they cannot win.

MCTS is already well established as a search technique for perfect information games and can be integrated into PIMC search in the obvious way, using MCTS to search each determinization. This combination has been investigated by a number of authors in in games such as Phantom Go [76], Phantom Chess (Kriegspiel) [77], and Klondike Solitaire [48] among others. Long et al [30] use MCTS with PIMC to try and address why PIMC is so successful despite the clear problems with the technique. In this thesis the issues with PIMC are addressed by developing new MCTS algorithms. The first question however, is to establish to what extent strategy fusion and non-locality are detrimental to the performance of PIMC. This question is investigated for the popular Chinese card game Dou Di Zhu (which is described in Chapter 4.2.1) by comparing algorithms which are allowed to "cheat" with PIMC and a variant of Expectimax search (defined in Section 3.3.2).

5.1.1 Balancing Simulations and Determinizations

This section presents work done to tune PIMC with MCTS for optimal performance in a particular game given a fixed computational budget (with results taken from three publications [50, 26, 27]). When integrating MCTS into PIMC, there are now two "Monte-Carlos", the Monte-Carlo sampling of determinizations used by PIMC and the Monte-Carlo sampling of simulated games used by MCTS. Let d denote the number of determinizations and s denote the number of MCTS simulations performed per determinization. The runtime of the algorithm is proportional to the product of these parameters ds, since the product corresponds to the total number of games played out across all determinizations and it can be reasonably assumed that the run-times of each play-out are independent and identically distributed random variables.

Since the ultimate aim of this work was to produce generic reusable AI for deployment in commercial games, the algorithm must return a decision within a fixed amount of time. Therefore the optimal ratio between determinizations and MCTS iterations per determinzations must be discovered for a particular game. For these initial experiments PIMC with UCT is used as a baseline algorithm to

investigate the effect of varying s and d. The results indicate that the balance of simulations and determinization is important and may differ between games. The methodology used to find this balance could easily be adapted to any game.

Choosing a representative set of deals for Dou Di Zhu

Although the strength of decisions made by each player has a significant effect on the outcome of a game of Dou Di Zhu, some random deals may favour one player over another, whereas others may be much more sensitive to the player's decisions. In an effort to reduce the variance of subsequent results and thus allow them to be compared more easily, a set of 1000 Dou Di Zhu deals was chosen as a representative sample for the remainder of the experiments in this chapter. The practice of specifying deck ordering in advance is common in Bridge and Whist tournaments between human players, to minimise the effect of luck when comparing players. It should be noted that experiments presented for Dou Di Zhu were qualitatively identical when a larger set of deals was used (see Section 5.2.4), suggesting that a set of 1000 deals is sufficiently large for comparing algorithms.

This set of deals was chosen such that when players use PIMC search with MCTS, the number of wins for a particular player is as close as possible to the mean number of wins for 1000 random deals. In order to choose such a set, the mean must first be determined. This was achieved by generating 100 sets of 1000 random Dou Di Zhu deals and for each deal a single game was played, using 50 determinizations and 250 MCTS iterations per determinization for each player. For each set, it was recorded how many of the 1000 games were won by player 1. Figure 5.1 shows a histogram of these numbers of wins. These results indicate that the number of wins appears to be normally distributed. The mean is $\mu = 433.47$ and the standard deviation is $\sigma = 16.27$ and so a 95% confidence interval for the mean number of wins for player 1 is $[433.37, 433.57]$. Therefore a set of deals for which player 1 (the landlord player) achieved exactly 433 wins was chosen as a representative set.

Varying Simulations and Determinizations for Dou Di Zhu

Now that a suitable set of deals has been chosen, the effect of varying the parameters d and s for PIMC with MCTS can be tested. Figures 5.2, 5.3 and 5.4 show the results of varying s and d for Dou Di Zhu. In these experiments, each combination of s and d is tested across all 1000 deals from the representative set. In each game, players 2 and 3 (the non-landlord players) use 40 determinizations and 250 UCT iterations per determinization, whereas player 1 uses d determinizations and s iterations per determinization, each game with a different value for d and/or s. For each combination of d and s, the number of games out of the 1000 won by player 1 is counted.

In this first experiment values for s are chosen from the set: $\{50, 100, 250, 500\}$. For each value of s, a number of values for d is tested, ranging from 1 to 100. Figure 5.2 shows the number of wins for each value of s. The results indicate

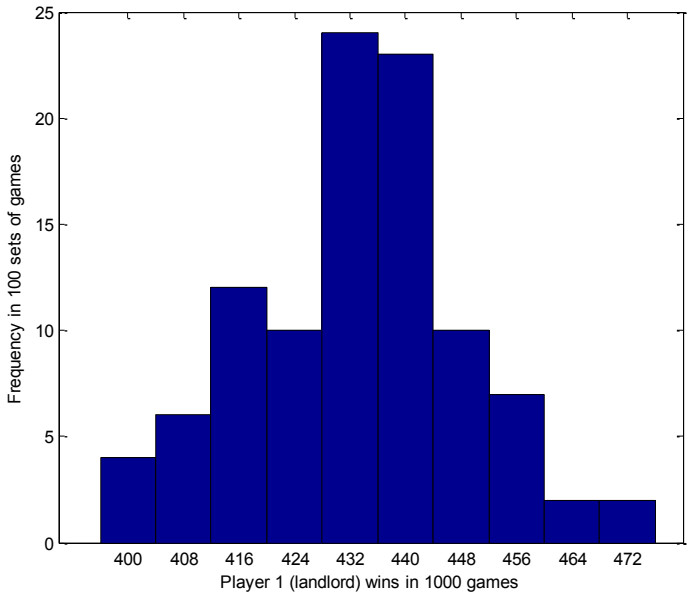

Figure 5.1: Histogram of win rates for the landlord player in 100 sets of 1000 Dou Di Zhu games. Each player used 50 determinizations with 250 MCTS iterations for each decision.

that playing strength increases rapidly with $d < 20$, with diminishing returns for $d \geq 20$. However, there seems to be slightly more benefit to increasing the number of determinizations beyond 30 when the number of UCT iterations is low. Next the effect of varying s whilst d remains fixed was tested. The conditions for this experiment were similar to those for the previous experiment, with the exception that values of d were chosen from the set: $\{5, 10, 25, 40\}$ and s varied from 25 to 1000. The results are plotted in Figure 5.3. For $s \leq 300$ the playing strength increases approximately logarithmically with the number of simulations, levelling off for $s > 300$.

Arguably the fairest comparison of the relative strength of different AI agents is to allocate them the same length of time (or the same number of CPU cycles) to make their decisions. The Dou Di Zhu implementation in $C\#$ used for experiments uses around one second to make a decision using 10000 MCTS iterations. Therefore using a budget of up to 15000 iteration is a good benchmark for a player that would return decisions quickly enough to play against in a commercial product without frustrating the user (accounting for the fact that a faster rate of iterations could be achieved with a well optimised implementation). It can also reasonable be assumed that the time taken for a single UCT iteration is roughly constant and the overall decision time is roughly linear in the number of iterations.

For PIMC with MCTS the total number of MCTS iterations across all determinizations is the product ds. Therefore in the final experiment four total iteration budgets $b = ds$ are fixed: $b \in \{2500, 5000, 10000, 15000\}$, and for each b, d is varied from 1 to 100, with $s = \frac{b}{d}$. As in the preceding two experiments, players 2 and 3 have $d = 40$, $s = 250$. Figure 5.4 plots the number of wins for player 1 for different values of d and a fixed b. Given the results of the preceding experiments, it is not surprising that performance is weaker when s or d is too small, nor that performance is somewhat independent of these parameters when both are sufficiently large. It is worth noting that for some iteration budgets, the performance of PIMC has a peak as the number of determinizations varies. This reinforces the need to tune the ratio of d to s for each games.

Varying Simulations and Determinizations for Lord of the Rings: The Confrontation

In this section PIMC with MCTS is applied to the game Lord of the Rings: The Confrontation (LOTR:C). A similar experiment to that in Section 5.1.1 was conducted to determine the correct balance of simulations and determinzations, but given the asymmetric nature of the game the experiment was conducted separately for each player, Light and Dark. Results of this experiment for LOTR:C are shown in Figure 5.5. Let $d \times s$ to refer to a player using PIMC with UCT using d determinizations and s iterations per determinization.

Contrary to the results for Dou Di Zhu, for LOTR:C the playing strength appears to worsen as the number of determinizations increases for a fixed value of ds. For instance, a Light 1×10000 player significantly outperforms a 40×250 player by 22.9%. Against a 40×250 player a 40×10000 player achieved win

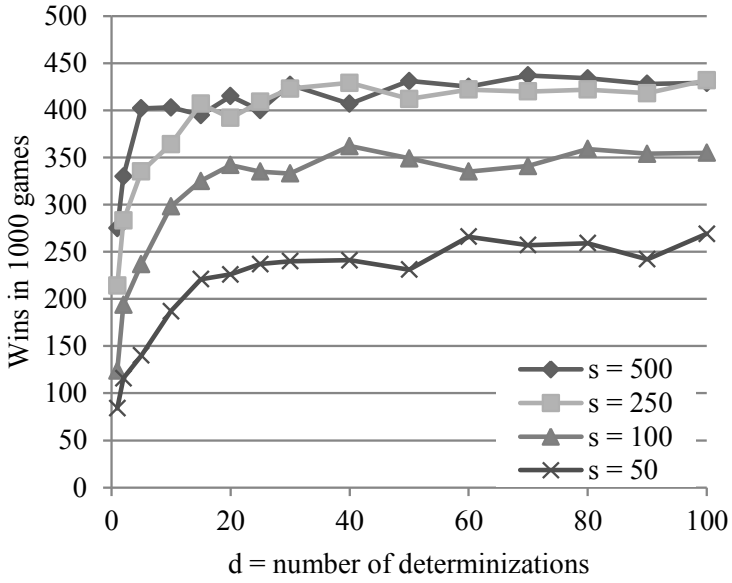

Figure 5.2: Plot of number of landlord (player 1) wins against number of determinizations, for fixed numbers of UCT iterations per determinization.

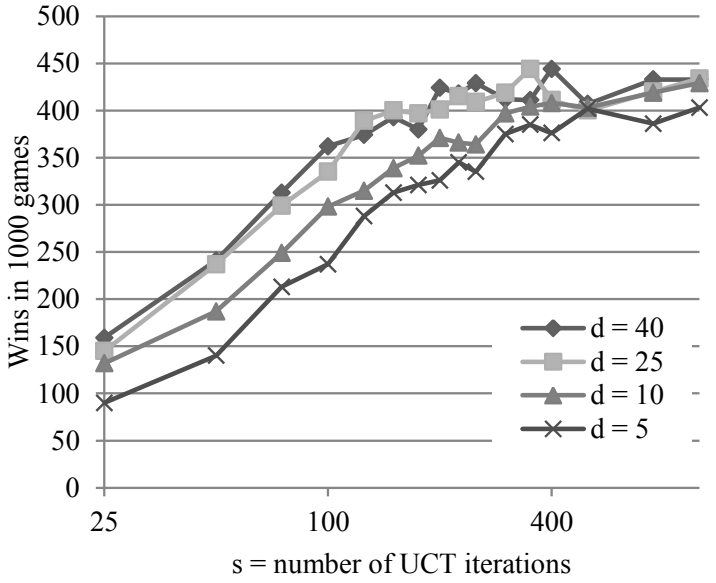

Figure 5.3: Plot of number of landlord (player 1) wins against number of UCT iterations per determinization, for fixed numbers of determinizations.

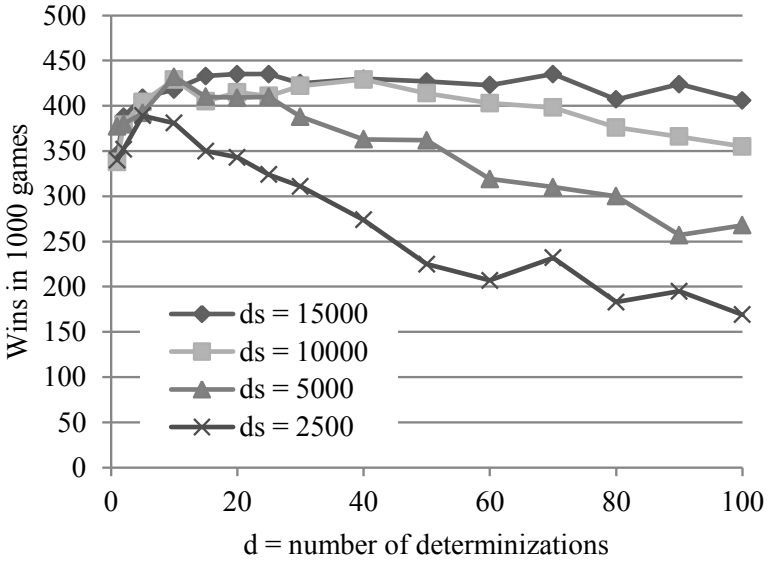

Figure 5.4: Plot of number of landlord (player 1) wins against number of determinizations, for a fixed number of UCT simulations divided equally among all determinizations.

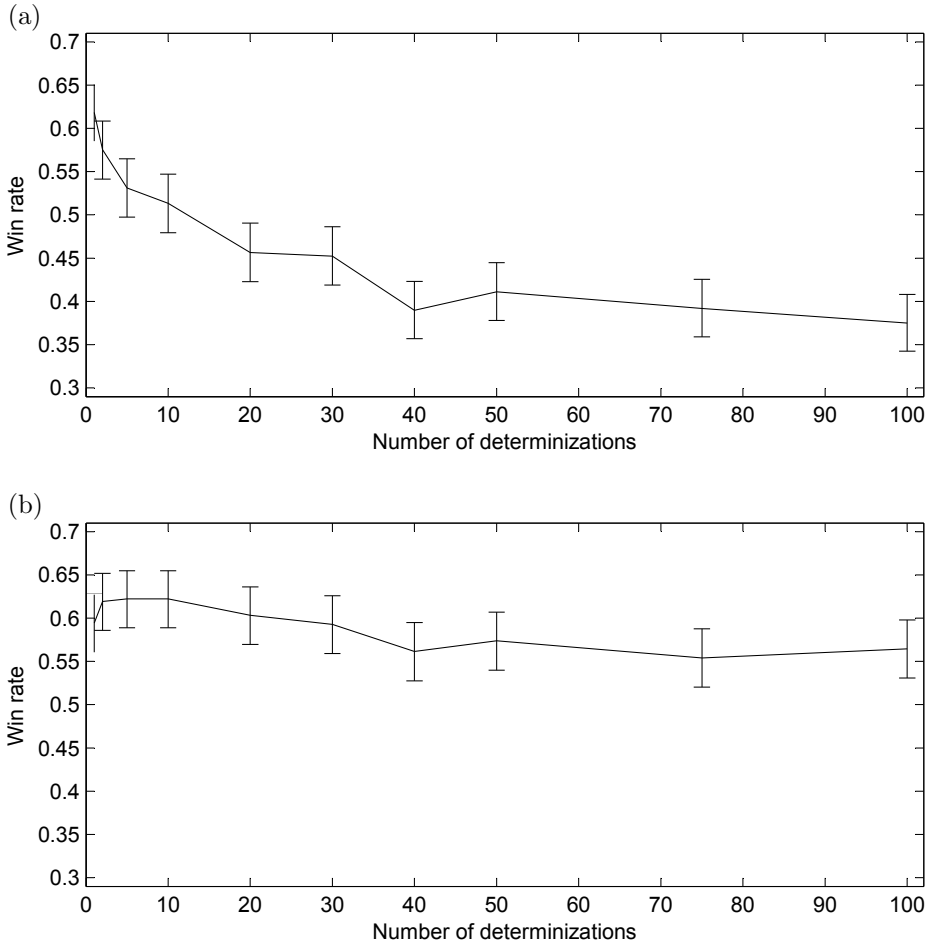

Figure 5.5: Results of the determinization balancing experiment for LOTR:C. Graph (a) shows the win rate for a Light player using PIMC with d determinizations and $\lfloor 10000/d \rfloor$ MCTS iterations per determinization, against a Dark opponent with 40 determinizations and $10000/40 = 250$ MCTS iterations per determinization. Graph (b) shows the same with Light and Dark exchanged (in particular, win rates are for the Dark player).

rates of 73.2% for Light and 83.0% for Dark, which exceed significantly the corresponding win rates for 1×10000. This is a consequence of the increased number of iterations per determinization, rather than the reduced number of determinizations: Naturally, the 40×10000 player also takes approximately 40 times longer to make each decision than a player using a total of 10000 iterations, so when ds is fixed at 10000, 1×10000 is the best setting for the Light player.

For the 1×10000 player, the average depth of the tree constructed from the initial game state is 8.6, and the average depth of a node is 4.0. For 40×250, the average tree depth is 4.1 and the average node depth is 2.4. Given that a single instance of combat in LOTR:C can account for five or more levels in the tree, searching to a depth of 4.1 is simply insufficient to make an informed decision. The effect of worsening playing strength as the number of determinizations is increased is more pronounced for the Light player. One possible reason for this is that Light's primary win condition (moving Frodo into Mordor) requires more long-term planning and thus deeper search than Dark's primary win condition (kill Frodo).

It can be concluded that in order to apply PIMC search with MCTS, it is always necessary to find the correct balance of simulations and determinizations since the best balance was different for both Dou Di Zhu and LOTR:C. More determinizations allows for a more diverse sample of possible states, but allows for less time to analyse each state given a fixed computational budget.

5.1.2 Quantifying the effect of strategy fusion and non-locality

The effects of strategy fusion and non-locality can be estimated by comparing PIMC search to a player that cheats to know the actual state of the game. If each determinization is the actual state of the game, then there is no strategy fusion since each determinization is the same and an action that is good in one determinization is good in all others. Similarly cheating is not prone to the effect of non-locality, since the determinizations used are always consistent with observed opponent play.

There are three advantages of a cheating player over PIMC search:

1. Lack of non-locality and strategy fusion
2. Inference (knowledge of information that could conceivably be inferred from the opponent decisions)
3. Clairvoyance (knowledge of information that could not possibly be inferred).

Quantifying the effect of each of these benefits should indicate which aspect of PIMC search offers the best potential for improvement. Clearly the third of these advantages is unattainable for a player that does not cheat. If the effect of knowing information that could be reasonably inferred is large, then opponent modelling and inference techniques could improve PIMC (for example

	Cheating	Determinized	Difference
Player 1	49.9%	43.0%	6.9%
Player 2	65.6%	57.0%	8.6%
Player 3	68.9%	57.0%	11.9%
Player 2 & 3	78.0%	57.0%	21.0%

Table 5.1: Playing strength of players with perfect versus imperfect information. Each row shows the win rate for the specified player(s) when they use cheating UCT or determinized UCT and all other players use determinized UCT.

through the biasing of determinization sampling). If the effect of strategy fusion is large, then developing algorithms which do not suffer from strategy fusion is a promising approach to improve upon PIMC. Experimental evidence presented in in this section suggests that the effect of the second is negligible for Dou Di Zhu (this may be different for other games) and so addressing the first is the most promising avenue to obtaining a strong imperfect information player.

Benefit of Cheating in Dou Di Zhu

In this experiment [50, 26] PIMC with UCT is tested with and without cheating in varous combinations. The number of wins (playing one game from each of the 1000 deals identified in Section 5.1.1) when each player uses PIMC with MCTS is used as a baseline and the increase in numbers of wins when the players are allowed to cheat is measured. Table 5.1 shows the results of this experiment. As expected the cheating player outperforms PIMC search which does not cheat. It is not surprising that the former outperforms the latter. However, by considering a simplified version of the game, it can be argued that the majority of this apparent advantage is not attributable to the difference in information available to the two agents, but is instead a consequence of problems with the method of determinization itself.

Algorithms for Mini Dou Di Zhu

The game tree for full Dou Di Zhu has a single chance node corresponding to the dealing of cards to each player. However, even after fixing the root player's own cards, the branching factor at this node is of the order 10^9 so searching this tree directly is impractical. Instead a simplified version of Dou Di Zhu called Mini Dou Di Zhu was used for experiments, which is small enough to be enumerable and solvable using minimax and expectimax algorithms. These algorithms are deterministic and solve games exactly, which means that the modelling assumptions behind the design of each algorithm can be compared rather than the ability of the algorithm to solve Mini Dou Di Zhu.

Thus two classes of algorithm are tested. Firstly PIMC search which analyses numerous determinizations separately. Then the Expectimax algorithm which constructs and searches a tree of information sets, avoiding the strategy fusion

problem with PIMC. In addition, cheating versions of the algorithms tested, which are allowed to observe the true state of the game and thus search the actual game tree.

Cheating minimax The minimax algorithm can easily search the entire depth of the Mini Dou Di Zhu game tree. Minimax is optimal against a minimax opponent for a perfect information game, but can occasionally make poor decisions against other types of opponent. For example, suppose that the minimax player has available two lines of play: one is a certain loss, while the other is a loss only if the opponent plays optimally from that point. Both lines have a minimax value of -1 so minimax chooses between them arbitrarily. However, if there is any possibility that the opponent will make a mistake (i.e. deviate from the minimax policy, which is not unlikely if the opponent is PIMC and does not cheat since the game does not have perfect information) then the second line is clearly the better choice. To solve this problem, minimax is modified with the following tie-breaking mechanism. Each state s is assigned a value $m_\varepsilon(s)$ by

$$m_\varepsilon(s) = \max_{a \in A(s)} -m_\varepsilon(f(s,a)) + \varepsilon \frac{\sum_{a \in A(s)} -m_\varepsilon(f(s,a))}{|A(s)|}$$

The first term is the standard Negamax formulation of the minimax value; the second term is proportional to the average value of playing a random action from state s. If two moves have the same minimax value, the tie will be broken by choosing the move that gives the opponent more opportunities to make a mistake. The constant ε must be small enough that if $m_0(s) < m_0(s')$ (where m_0 denotes the standard minimax value) then $m_\varepsilon(s) < m_\varepsilon(s')$, so that actions that maximise m_ε also maximise m; in other words, the set of actions identified as optimal by m_ε is a subset of those identified as optimal by m.

PIMC Minimax A determinization approach similar to the approach Ginsberg [16] uses for Bridge is applied, where PIMC is combined with an algorithm to solve the perfect information game (minimax search in this case). In Mini Dou Di Zhu, the maximum number of states in an information set is 66 (calculated through enumeration), so it is feasible to iterate over all possible determinizations. Each determinization is small enough to be solved exactly by the minimax algorithm. The results of the individual minimax searches are combined by weighted majority voting: the number of votes for an action is the sum of probabilities of determinizations in which that action is chosen by minimax and the action selected by the agent is one for which this sum is maximal. The determinization process is illustrated in Figure 5.6(a). There is an initial branch for each possible determinization for the current information set, each labelled with its associated probability. Each determinization fixes the structure of the tree that is then searched by minimax.

Information set expectimax This algorithm is as an analogue to the ISMCTS algorithm (introduced later in this chapter), since instead of searching deter-

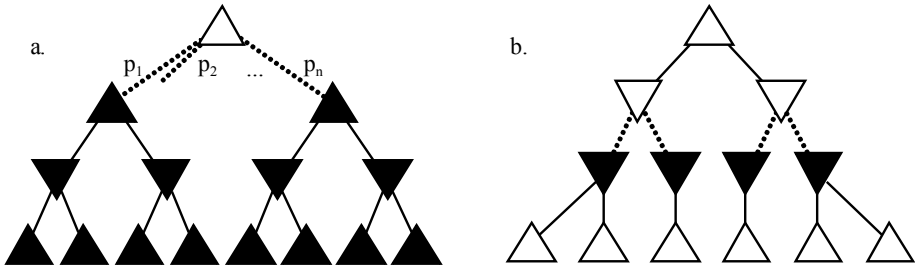

Figure 5.6: Illustration of search trees for (a) determinization and (b) information set expectimax. White triangles correspond to information sets, and black triangles to individual states. Triangles pointing upwards denote nodes belonging to the root (maximising) player, and triangles pointing downwards to the opponent (minimising) player. Solid lines denote decisions, whereas dashed lines denote decomposition of information sets into their constituent states. All information sets are from the point of view of the root player.

minized trees of states, this algorithm searches a tree of information sets. This avoid the strategy fusion problem with PIMC search since a single value is computed for each action in an information set, rather than different values for each state within an information set. To search a tree of information sets in Mini Dou Di Zhu the Expectimax search algorithm [78] is used. As discussed in Section 3.3.2, Expectimax extends the minimax algorithm to game trees containing chance nodes: the value of a chance node is the expected value of choosing one of its children at random. For trees of information sets the opponent's decision nodes are treated as chance nodes (branching for the states in the current information set) followed by perfect information decision nodes. This is illustrated in Figure 5.6(b). Each non-terminal information set $[s]_j$ is assigned a value $v_j([s]_j)$ recursively by

$$v_j([s]_j) = \begin{cases} \max_{a \in A([s]_j)} v_j([f(s,a)]_j), & \text{if } \rho([s]_j) = j \\ \mathbb{E}_{s \in [s]_j} \min_{a \in A(s)} v_j([f(s,a)]_j), & \text{if } \rho([s]_j) = j \end{cases}$$

Terminal information sets assigned values of ± 1 for wins and losses in the usual way. The agent selects a move to maximise the value of the resulting information set.

Inference for Mini Dou Di Zhu Finally, in order to measure how much of the benefit of cheating is due to inference (and clairvoyance) as opposed to lack of strategy fusion, versions of each algorithm are tested which use an opponent model. In games of hidden information, it is often possible to infer information by observing the actions of the other players, according to some model of the other players decision processes. This type of inference has frequently been applied to the game of poker [116, 89], but also to other games such as

Scrabble [90] and the card game Skat [49] which has similarities to Dou Di Zhu. Performing inference with MCTS is discussed in more detail in Chapter 7.

In Mini Dou Di Zhu, inference can be performed by applying an opponent model to all states in the current information set and comparing the observed move from the current information set with the opponent model's choice from each state: if the moves for a particular state do not match, it can be concluded that state is not the true state of the game. where the probability of playing a move given a state is 0 or 1, similar to Richards and Amir [90] (which is the case since the minimax and expectimax players tested for Mini Dou Di Zhu are deterministic). This type of inference requires consideration of all states in the current information set, which is infeasible for the full game of Dou Di Zhu, but tractable for Mini Dou Dhi Zhu since there are only 8832 possible deals.

Effect of Cheating in Mini Dou Di Zhu

In this section an experiment is performed to compare the playing strength of the different algorithms described earlier for Mini Dou Di Zhu, when they cheat compared to when they do not cheat. Specifically, the agents are cheating minimax, determinized minimax and information set Expectimax; for the latter two, variants are tested with no inference model, Bayesian inference with an incorrect opponent model and Bayesian inference with the correct opponent model. Here the "correct" opponent model uses exactly the same algorithm as the opponent, whereas the "incorrect" model uses a different algorithm to the opponent. The former can be considered a best case for the effectiveness of inference, whereas the latter is a more realistic test of its effectiveness when an exact opponent model is not known.

In each case all 8832 deals are iterated through, playing a single game for each combination of agents (a single game suffices as all agents are deterministic). The measure of a particular agent's strength against some opponent is the probability that it wins a randomly dealt game, which is obtained by summing the probabilities of those deals from the 8832 that it wins. The results of this experiment are shown in Figure 5.7, indicating that the win rate for cheating minimax is approximately 40% greater than that of determinized minimax although the exact value depends on the opponent type. It should come as no surprise that cheating outperforms determinization. The former has access to information that the latter does not and so can make more informed decisions and better anticipate its opponent's responses.

Effect of Inference in Mini Dou Di Zhu

Inference improves the strength of determinized minimax, increasing the win rate by around 10%, or 23% with a perfect opponent model. However, the performance of determinized minimax with inference still falls short of expectimax without inference. The determinized minimax agent with inference and the correct opponent model correctly identifies the actual state of the game (i.e. assigns the actual state probability 1 and all other states probability 0) by the

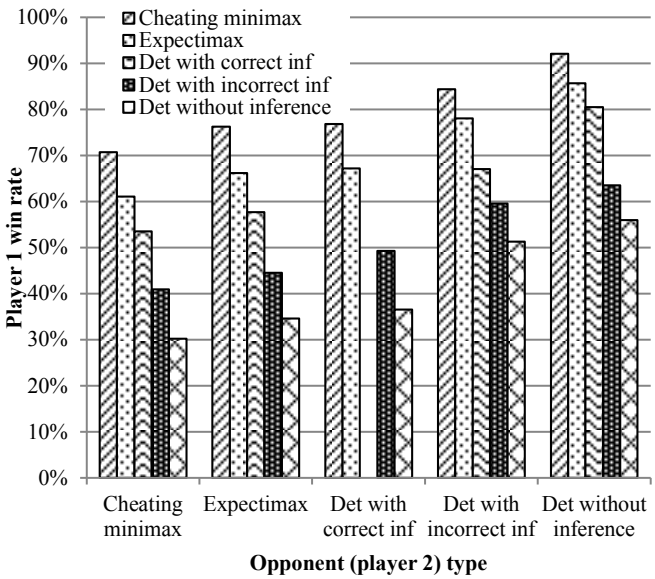

Figure 5.7: Performance of several agents in Mini Dou Di Zhu. For each group of bars, player 2 uses the algorithm specified in the x-axis label; for each bar within a group, player 1 uses the algorithm specified by the legend. The bars themselves represent the win rate over all deals for player 1. Determinization with correct inference was not tested against itself: while possible, it is not straightforward to implement this in a way that does not require infinite nesting of opponent models [26]. Error bars are not included since the players are deterministic and win rates are exact across all deals.

fourth turn in 48% of deals against an expectimax opponent; by the end of the game, this proportion of deals increases to 74%. Inference significantly improves the performance of determinized minimax; however the same cannot be said for expectimax.

A variant of the expectimax player was tested in which the probabilities used in calculating expected values at opponent nodes are determined by Bayesian inference. The resulting increase in playing strength is negligible (less than 0.2%). (Since these results are almost identical to those for expectimax without inference, they are omitted from Figure 5.7.) One explanation could be that inferred information is generally not helpful in Mini Dou Di Zhu and the apparent benefit of inference in determinized minimax arises by reducing the number of determinizations to be searched, thus reducing the variance among the minimax results and ameliorating the effects of strategy fusion.

Expectimax is a significant improvement over determinized minimax, outperforming it by around 30% and achieving a win rate around 10% lower than that of cheating minimax. This suggests that contrary to the intuition that knowing hidden information should be greatly beneficial, approximately three quarters of the apparent benefit of cheating in Mini Dou Di Zhu can actually be attributed to the effects of strategy fusion, from which expectimax does not suffer.

5.2 Information Set Monte Carlo Tree Search (ISMCTS)

By studying the performance of deterministic algorithms (cheating minimax, determinized minimax and expectimax) on Mini Dou Di Zhu, it appears that a large proportion of the apparent benefit of cheating has less to do with gaining access to hidden information and more to do with overcoming the inherent shortcomings of determinization. Furthermore for mini Dou Di Zhu it appears inference has little effect on the strength of an agent that does not use determinization. Giving determinized minimax knowledge about the state of the game provided little benefit. However for mini Dou Di Zhu, the algorithm which does not suffer from strategy fusion performed well. Therefore it can be concluded that much of the benefit of cheating for this game is due to a lack of strategy fusion and that the potential benefit of inference is small for Dou Di Zhu. For mini Dou Di Zhu it has been shown that strategy fusion is a significant problem for PIMC. In this section *Information Set Monte Carlo Tree Search* (ISMCTS) is introduced as an improvement upon PIMC with MCTS, designed not to suffer the effects of strategy fusion.

When applying PIMC with MCTS, the strategy fusion problem arises since the values of actions are learnt in the context of specific states. One solution to this is to learn action values in the context of particular information sets instead. These values can be stored in a single decision tree, where the nodes correspond to information sets of the decision making player. However, if the same de-

terminization is searched on many consecutive iterations, the values learnt will have a strong amount of local bias towards actions which are good in the current determinzation. Therefore ISMCTS is introduced which does not search a tree of states, rather a tree where nodes correspond to information sets. Since building a tree with a unique node for every unique information set is intractable in many large games, the ISMCTS algorithm utilizes different techniques to group information sets into a single node in order to reduce the complexity of the tree. In addition the ISMCTS algorithm uses a new determinization on each MCTS iteration in order to ensure an unbiased mix of determinzations. The ISMCTS algorithm is defined in Section 5.2.3, but first the issues of subset armed bandits and chance nodes must be addressed.

ISMCTS offers numerous advantages over PIMC search with MCTS in addition to removing the effect of strategy fusion. PIMC does not transfer knowledge between determinization except in the final step of making a decision. ISMCTS constructs a single tree, so allowing information to be shared as the search progresses. ISMCTS also removes the need to find the correct balance between determinizations and simulations. ISMCTS does not address any of the other shortcomings of PIMC search particularly non-locality which is addressed in Chapter 7.

5.2.1 Subset-armed Bandits

One issue with determinization is with decisions for players other than the observer. Another player may be able to distinguish between states in an information set and may wish to make different decisions in each case. This means that in an information set tree at opponent nodes, the utility of each action may depend on the current determinization. Furthermore in many games an observer cannot predict the set of actions that will be available to another observer in a particular information set. This leads to an action selection problem at opponent information sets during tree search where a different set of actions will be available depending on the current determinization.

One solution to this problem would be to search each tree corresponding to each possible opponent information set independently (in addition to a players own information set). Within each of these trees, the set of available actions would remain constant and so would the utility associated with each action. For many games of imperfect information there are a very large amount of possible determinizations which would make such an approach infeasible. This is because bandit algorithms applied to tree search rely on many trials of each state in the tree in order to discover the best actions to choose. In practice it is preferable to group together determinizations in which the opponent decision is likely to be correlated. This allows the opponent to make different decisions in some different states, but also allows each group to receive a reasonable amount of trials. This would still lead to the overlapping of action sets which defines a different problem to the multi-armed bandit problem.

Therefore ISMCTS introduces a new issue not present with PIMC search when only one player's information sets are searched; the *subset armed bandit*

problem. In determinizations, the game is perfect information so at opponent decisions it is known exactly what state the opponent is in. However with ISMCTS, the same opponent decision node is used for many possible opponent information sets. For example in a card game, at the decision making player's nodes it is known what information set the player will be in because the player can see their cards. At opponent's decision nodes however, there may be many possible information sets corresponding to the different deals the opponent has in each determinization. Therefore the set of legal actions at that decision node may be different on each iteration, due to the opponent having a different set of cards in hand.

The subset armed bandit problem

The subset armed bandit problem is defined as follows:

Definition 11. A **k -Armed Subset-Armed Bandit Problem** consists of a k-Armed Multi-Armed Bandit Problem and a set of n random variables denoted A_n. If $I := \{1, \ldots, k\}$ and $\mathcal{P}(I)$ is the power set of I then each $A_n \in \mathcal{P}(I)$ and on trial n only arms in the set A_n are available for selection. Random variables A_n are independent and identically distributed according to some distribution where $\mathbb{P}(A_n = \emptyset) = 0$ (which ensures that at least one arm is always available for selection).

A policy for the multi armed bandit problem can be applied to the subset armed bandit problem by treating each subset as a regular multi armed bandit problem and applying the policy independently in each subset. However this technique does not exploit the fact that the distribution of an arm may not depend on the subset of arms available. Instead each arm must be considered separately for each subset, but this leads to a large increase in the number of arms and is not tractable for ISMCTS.

Handling subset armed bandits

Performing tree search using a method similar to that described in the previous section may not be desirable in practice. The first issue is that a set of available actions may not uniquely correspond to a state and that the optimal arm may also depend on the state. If the number of possible states is large then it may not be possible in practice to measure the utility of each action in each state. Therefore it is convenient to assume that the optimal action remains unchanged amongst all states which produce the same set of legal actions. In practice this assumption does not cause problems for many games since it is often the case that the utility of an action is somewhat consistent across states in an information set (which happens to be the property that PIMC relies upon).

The second issue is that if sets of available actions overlap, then it is possible that the value of an action in two different sets is the same. More specifically, it may be that by averaging the utility of an action over two different sets does not change which action is the optimal action in each set. By spending time

measuring the utility of each action independently in both sets, more time is spent exploring than is necessary. It would therefore be desirable to perform only one exploration phase per action, but this would result in a sub-optimal arm being exploited if averaging the utility across overlapping sets of actions changed the optimality of arms.

Therefore the subset armed bandit problem introduces a few theoretical problems with ISMCTS. Firstly, the value of an action for an opponent may be different for each opponent information set. There may be a large number of opponent information sets however (for example all possible opponent deals in a card game), so learning the unique value for each opponent information set can be intractable. For games with smaller numbers of information sets this is possible (and discussed later in Chapter 7). In other cases, it is assumed the average value across all opponent information sets can be used to measure the utility of an opponent action. This leads to a problem which is the inverse of strategy fusion, a *strategy fission* where the opponent is assumed not to be able to choose different actions depending on which information set they are in. In Chapter 6 it is shown how this strategy fission can be avoided in the case of partially observable actions.

Finally the subset armed bandit problem can lead to unmanageable branching factors in games where the number of possible legal moves is large. For example in Dou Di Zhu, moves consist of combinations of cards in hand and substituting one card for another can lead to a large number of new unique moves. At opponent nodes in ISMCTS, many iterations will be expanding a new unseen legal move. In addition if using the UCB1 algorithm, rarely seen moves will have a large exploration urgency. This can be alleviated by growing the exploration urgency in proportion to the number of times a move was available for selection. Denote by $\overline{T}_i(n)$ the number of times arm i was available during the first n trials. If $\overline{x}_i$ is the average reward received from arm i then select randomly amongst any arms for which $T_i(n) = 0$, and otherwise randomly amongst any arm for which

$$\overline{x}_i + C\sqrt{\frac{ln(\overline{T}_i(n))}{T_i(n)}} \tag{5.1}$$

is maximal, for some constant C. This is equivalent to the UCB algorithm except the number of times an arm was available is substituted for number of trials. This means that the exploration bonus for an arm only grows whilst it is available for selection, meaning that arms that are infrequently available will not be over-explored.

5.2.2 Handling Chance Events

As defined in Section 2.1 chance events (or stochastic outcomes) are actions performed randomly by the environment player. Handling of chance events is not a primary focus of this thesis. However, chance nodes do occur under certain circumstances in several of the test domains (see Section 4.2), so they cannot

be ignored completely. Note that in the games studied in this thesis chance nodes typically have a small number of possible outcomes. Technically, card games include a chance event with a large number of outcomes corresponding to shuffling and dealing a deck of cards at the beginning of the game, but since this occurs before any player has made a decision it never occurs as a chance node in the search tree. Therefore it was sufficient to simply include chance nodes in MCTS trees and sample outcomes evenly.

Consider a chance node with k branches, each of which is equally likely. To ensure that each branch is explored approximately equally, the first k visits select all outcomes in a random permutation, the second k visits select all outcomes in another random permutation, and so on. This is almost trivial to implement in UCT: since UCB with random tie-breaking is used for action selection, it suffices to treat the environment player as a decision-making agent who has perfect information and receives a reward of zero for all terminal states. The UCB exploration term then ensures that the branches are visited in the manner described above. This method only worked for uniformly random chance nodes, but could be adapted to the non-uniform situation by changing the selection policy. This approach was applied to most of the domains in this thesis with the exception of Backgammon, where future dice rolls are treated as hidden information (expressed through a seeded random number generator), so that the same rolls occur on the same determinization.

5.2.3 The ISMCTS Algorithm

Now that the issues of chance node and subset armed bandits have been addressed, the ISMCTS algorithm can be defined. In summary, ISMCTS aims to overcome the problems associated with the PIMC search, by searching a single tree whose nodes correspond to information sets rather than states. In ISMCTS, nodes in the tree correspond to information sets from the root player's point of view, and edges correspond to actions (i.e. moves from the point of view of the player who plays them). The correspondence between nodes and information sets is not one-one: partially observable opponent moves that are indistinguishable to the root player have separate edges in the tree, and thus the resulting information set has several nodes in the tree. Partially observable actions are treated in depth in Chapter 6.

The nodes in an ISMCTS tree represent unique histories from the point of view of the root player. This means that nodes are distinguished only by the observation the root player makes of the actions which lead to the node, so that some nodes may correspond to many different information sets in the game which are indistinguishable to the root player (but appear distinct to another player). For example in a card game observing a sequence of cards being played leads to a unique node in the ISMCTS tree, but there may be many possible opponent information sets with this history of played cards (corresponding to the different cards the opponent may hold). This is similar to the approach taken by Nau et al [81] in Bridge where Minimax search was restricted to branches corresponding to certain tactics or strategies in order to reduce branching factor

(with ISMCTS restricting transitions to those observable by the root player).

Grouping together information sets which are indistinguishable to the root player in this way (using only the observation of actions) dramatically reduces the number of nodes in the search tree (in contrast to having a unique node for every unique information set). This allows the ISMCTS algorithm to learn faster by visiting opponent nodes more frequently and also avoids the need to keep track of information sets which reduces the complexity of the algorithm. The drawback of this approach is that the opponent model is weaker, since the algorithm cannot learn a different policy for different opponent information sets and introduces the issues of subset armed bandits which is dealt with in this Chapter. In practice this will not be a problem in a game where the best actions do not depend on the information set. One example is a card game where playing a specific card which wins the game, the other cards in the hand do not matter just the presence or absence of the winning card. In games such as The Resistance, the best strategy depends on a piece of hidden information (the secret role of the opponent) so grouping information sets in this manner is detrimental to the performance of the algorithm. This problem is dealt with in Chapter 7 but at the cost of massively increasing the number of nodes in the ISMCTS tree (and slowing down the learning rate).

In order to help illustrate the tree structure used by ISMCTS, Figure 5.8 shows a game tree for a simple 1-player game of imperfect information. The root information set contains two states, x and y. The player first selects one of two actions, a_1 or a_2. Selecting a_2 yields an immediate reward of $+0.5$ and ends the game. If the player instead selects a_1, they must then select an action a_3 or a_4. If the game began in state x, then a_3 and a_4 lead to rewards of -1 and $+1$ respectively (this information being revealed by means of environment action $e_{x,3}$ or $e_{x,4}$); if the game began in state y then the rewards are interchanged.

If states x and y are equally likely, action a_1 has an Expectimax value of 0: upon choosing a_1, both a_3 and a_4 have an Expectimax value of 0. Thus the optimal action from the root is a_2. However, a determinizing player searches trees corresponding to each state x and y individually and assigns a_1 a minimax value of $+1$ in each (by assuming that the correct choice of a_3 or a_4 can always be made), thus believing a_1 to be optimal. This is an example of strategy fusion.

Figure 5.9 shows the tree searched by ISMCTS for this game. In this case each node is in one-one correspondence with an information set. After a sufficiently large number of iterations the algorithm assigns each environment node an expected value of 0 and thus assigns the same value to action a_1, thus overcoming strategy fusion and correctly identifying a_2 as the optimal move.

Figure 5.10 shows a game tree for a more complex, 2-player game. The game starts in one of three states: x, y or z. These states are distinguishable to player 2 but not to player 1. Player 1 first selects an action a_1 or a_2. If he chooses a_1, player 2 then selects an action b_1, b_2 or b_3. However only two of these actions are available, and which two depends on the initial state. Player 1 then selects a_3 or a_4, and both players receive rewards as shown. Note that if player 2 chooses b_1 or b_3 then the rewards do not depend on the initial state, but if player 2 chooses b_2 then the rewards do depend on the initial state.

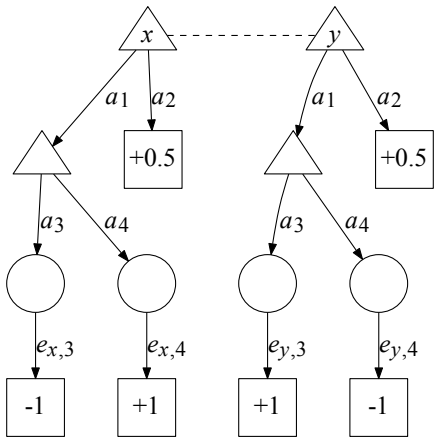

Figure 5.8: An example game tree for a simple 1-player game. Nodes represent game states. Nodes shaped $\triangle$ denote player 1 decision states, $\bigcirc$ environment states, and $\square$ terminal states labelled with reward values for player 1. Non-terminal nodes in corresponding positions in the x and y sub-trees are in the same player 1 information set; this is shown by a dashed line for the root nodes. Adapted from [30, Figure 1].

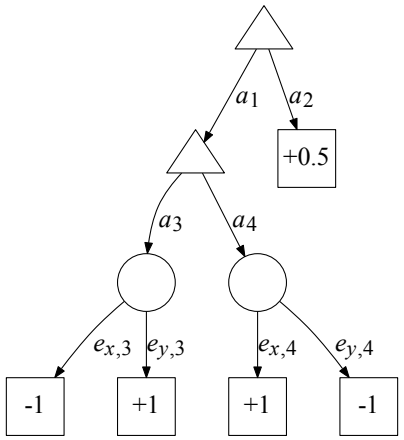

Figure 5.9: An information set search tree for the game shown in Figure 5.8. Here nodes shaped $\triangle$ denote information sets where player 1 is both the observer and the player about to act.

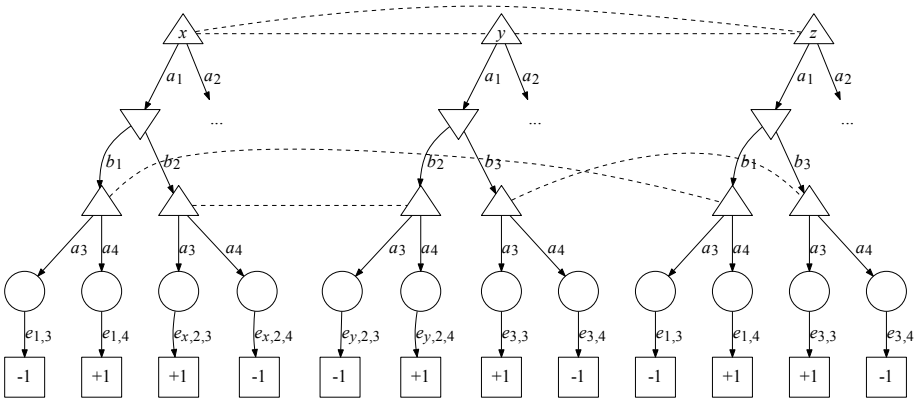

Figure 5.10: An example game tree for a simple 2-player game. Nodes shaped $\triangle$ denote player 1 decision states, $\triangledown$ player 2 decision states, $\bigcirc$ environment states, and $\square$ terminal states labelled with reward values for player 1 (the game is zero-sum, so player 2's 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. her information sets are singletons.

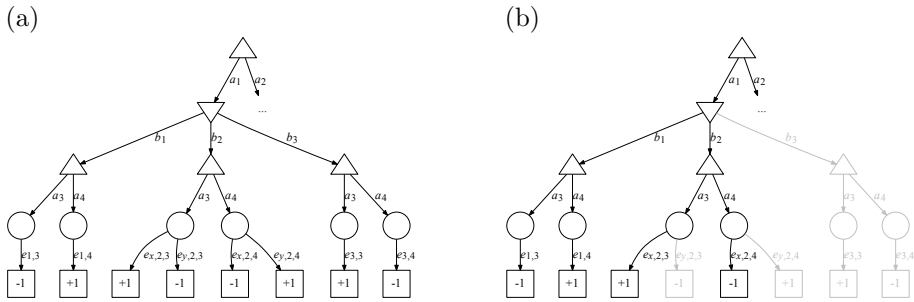

Figure 5.11: An information set search tree for the game shown in Figure 5.10. (a) shows the entire tree; (b) shows the restriction of the tree to determinization x.

Figure 5.11 (a) shows the tree searched by ISMCTS for this game. For an information set $[s]^{\sim_i}$ where the observer is not the player about to act, i.e. $\rho([s]^{\sim_i}) \neq i$, the set $A(s)$ of available actions can differ for different states $s \in [s]^{\sim_i}$. The set of legal actions may depend on information to which another player does not have access.

When searching trees of information sets, this creates a problem at opponent nodes. There must be a branch for every action that can possibly be available from that information set; this is illustrated in Figure 5.11 (a), where the opponent decision node has branches for all three actions b_1, b_2, b_3 even though only two of those three actions are available in each state $f(x, a_1), f(y, a_1), f(z, a_1)$ in the corresponding player 1 information set. However, the exploitation and exploration of actions must be balanced with how likely those actions are to be available. For example, it is desirable to avoid over-exploiting an action that is a certain win for the opponent but is only available with probability $\frac{1}{100}$ (i.e. in only one of 100 states in the information set).

ISMCTS addresses this problem, since at the beginning of each iteration a random determinization is sampled, which restricts that iteration to those regions of the information set tree that are consistent with that determinization. Thus the branches at opponent nodes are available for selection precisely as often as a determinization is chosen in which the corresponding action is available. In other words, the probability of an action being available for selection on a given iteration is precisely the probability of sampling a determinization in which that action is available. The set of actions available at an opponent node can differ between visits to that node, and thus action selection is a subset-armed bandit problem. Figure 5.11 (b) demonstrates such a restriction of the search tree shown in Figure 5.11 (a).

Consider the example tree in Figure 5.11 (b). Note that the restricted tree is never explicitly constructed, but the tree policy is restricted as it descends the tree by means of the determinization d. In turn, d is updated as the tree is descended by applying the selected actions. Otherwise, selection works as in plain UCT. Suppose that in determinization x the sequence of actions selected is $a_1, b_2, a_3, e_{x,2,3}$. Let us identify each of the visited nodes with its incoming action (i.e. the label of the incoming edge). At nodes $e_{x,2,3}, a_3, b_2, a_1$ and the root, the visit count $n(v)$ and total reward $r(v)$ is updated as usual. For these nodes and for all siblings that were also available for selection, i.e. including nodes a_4 and b_1 but *not* nodes b_3 and $e_{y,2,3}$, the availability count $n'(v)$ is incremented by 1. The availability count replaces the parent node's visit count in the UCB formula in order to adapt UCB to the subset-armed bandit problem, as discussed in Section 5.2.1.

Now that each component of ISMCTS has been explained, detailed pseudo-code is given in Algorithm 2 (page 86) with enough detail to enable a reader to implement the algorithm, whilst high-level pseudo-code for the SO-ISMCTS algorithm is presented in Algorithm 3 (page 87). In this and other pseudo-code in this thesis, it is assumed that player 1 is conducting the search. The pseudo-code in Algorithm 3 does not specify which bandit algorithm is used during selection, however the experiments in this thesis (and in Algorithm 2) UCB1

is used (modified for subset-armed bandits as described in Section 5.2.1). The following notation is used in this pseudo-code:

- $c(v)$ = children of node v
- $a(v)$ = incoming action at node v
- $n(v)$ = visit count for node v
- $n'(v)$ = availability count for node v
- $r(v)$ = total reward for node v
- $c(v,d) = \{u \in c(v) \ : \ a(u) \in A(d)\}$, the children of v compatible with determinization d
- $u(v,d) = \{a \in A(d) \ : \ \nexists c \in c(v,d) \text{ with } a(c) = a\}$, 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

The idea of constructing trees of information sets and sampling determinizations to restrict the region to be searched is similar to the Partially Observable UCT (PO-UCT) approach of Silver and Veness [82], although PO-UCT operates on the domain of partially observable Markov decision problems (i.e. 1-player games of imperfect information) rather than adversarial games. Schäfer [75] also applied an information set tree approach for the game Skat using the UCB1 algorithm for selection. The information sets are from the point of view of the player about to play, rather than from the point of view of one player as in ISMCTS. The fact that other authors have investigated single tree approaches, combined with the wide range of good experimental results (including new results in this thesis) suggest that ISMCTS is a significant advance beyond PIMC.

5.2.4 Evaluating ISMCTS

The principles behind the design of the ISMCTS algorithm have been discussed and the algorithm has been defined. This section presents empirical evaluations of ISMCTS for several domains. Many of the games ISMCTS is applied to in this thesis involve more than two players and some involve coalitions. Non zero-sum games and multi-player games introduce several new algorithm design problems (See Section 3.2.5) which are beyond the scope of this thesis. All ISMCTS trees are Max^n trees [117], although it is in theory possible to integrate different solution concepts for multi-player games into ISMCTS.

Mini Dou Di Zhu

Firstly ISMCTS is evaluated for the game Mini Dou Dhi Zhu, where the experiment [26] presented earlier (Section 5.1.2) motivated the design of ISMCTS. To test the effectiveness of ISMCTS, it was played against PIMC with UCT. In all

cases player 2 used PIMC with 20 determinizations and 200 UCT iterations per determinization. Each of the 8832 deals was played 10 times with player 1 using PIMC with 20 determinizations and 200 UCT iterations per determinization and 10 times with player 1 using ISMCTS with 4000 iterations. ISMCTS performed better with a 67.2% win rate versus 62.7% for PIMC. Clearly ISMCTS was effective in addressing the strategy fusion issue present in mini Dou Di Zhu. The next section investigates whether ISMCTS outperforms PIMC in the full version of the game.

Dou Di Zhu

Experiments were run to determine the amount of exploration that should be performed (i.e. the value of C in Equation 5.1) and the number of iterations required for good performance with ISMCTS. Each of the 1000 selected deals for Dou Di Zhu was played 5 times with the landlord player as ISMCTS, an exploration constant of 0.44 and varying numbers of iterations (between 500 and 30000) against PIMC with UCT opponents using 40 trees and 250 iterations per tree. The results indicated that the playing strength of ISMCTS increased up to 10000 iterations where it achieved a win rate of 40.8%. Increasing the number of iterations further had no significant effect on playing strength. Similarly each deal was played 5 times with the landlord player as ISMCTS using 10000 iterations and varying values for the UCT exploration constant (between 0.1 and 3). The results indicated that the algorithm performs poorly with exploration less than 0.5 and achieves best performance with exploration greater than 1. Increasing the exploration beyond 1 had little effect on playing strength.

To measure the difference between cheating UCT, PIMC with UCT and information set UCT, each of the 1000 deals of Dou Di Zhu was played 40 times with the landlord as cheating UCT, determinized UCT (PIMC with UCT) and ISMCTS. In all cases 40 trees with 250 iterations per tree or 10000 iterations in total were used and the exploration constant for information set UCT was chosen to be 1.0. The opponents used determinized UCT. First of all this data was used to calculate the average win rate for each player as the landlord. The results indicate that there is no significant difference in playing strength between ISMCTS (42.0%) and determinized UCT (42.4%). Cheating UCT was much stronger, achieving a win rate of 54.0%.

The original aim of developing ISMCTS was to overcome strategy fusion difficulties in the deals where the cheating player's advantage is largest. To investigate whether this has been achieved, the average win rate for each deal was calculated for each player type. Then for each deal the difference in win rate between cheating UCT and determinized UCT was calculated. By looking only at deals in which this difference is above a certain threshold, it is possible to compare the performance of information set UCT and determinized UCT in the deals where knowing the hidden information is most beneficial to the cheating player.

Similarly the difference between ISMCTS and determinized UCT can be compared for the deals in which knowing the hidden information is not benefi-

Threshold	PIMC win rate	ISMCTS win rate	Number of deals
None	42.4%	42.0%	1000
$< 0\%$	53.0%	45.2%	133
$= 0\%$	44.9%	43.3%	138
$> 0\%$	40.0%	41.1%	729
$> 25\%$	31.6%	38.4%	166
$> 50\%$	20.4%	35.0%	12
$> 75\%$	15.0%	95.0%	1

Table 5.2: Win rates for PIMC and ISMCTS. Each row shows win rates for a subset of the 1000 initial Dou Di Zhu deals, filtered by the difference in win rate between cheating PIMC and PIMC (Threshold column).

cial. The results of this experiment are presented in Table 5.2. For deals where the threshold in Table 5.2 is less than or equal to zero, cheating UCT does not outperform determinized UCT and the advantage of knowing the opponent's cards is not significant. In this situation ISMCTS offers no advantage and performs slightly worse than determinized UCT. When the threshold is greater than zero there is an advantage to knowing opponent cards and ISMCTS is significantly stronger than determinized UCT. Furthermore, as the gap between cheating UCT and determinized UCT increases, the gap between ISMCTS and determinized UCT increases also.

It should be noted that the structure of the trees searched by determinized UCT and cheating UCT are the same on average (this is discussed further in the next section). The most significant differences between the two are the access to hidden information and the consistency due to each UCT tree in the cheating player's ensemble (of 40 trees) corresponding to the same perfect information game. In deals where the cheating UCT player performed better than determinized UCT, and hence where hidden information and consistency in decision making had some impact, it was observed that ISMCTS performed better than determinized UCT. Since ISMCTS has no access to hidden information, this would suggest that the single tree approach is providing some of the same benefit the cheating ensemble player gains through consistency of the UCT trees searched. Indeed the results obtained for Mini Dou Di Zhu suggest that hidden information is not often important in Dou Di Zhu and it is a highly unusual feature of this game, that knowledge of information often has little impact on players that use determinization.

Since the cards dealt are a significant deciding factor in the outcome of a game of Dou Di Zhu, the observed results may have been influenced by the small sample size of 1000 deals. This experiment was repeated with a larger sample of 5000 new randomly chosen deals, where each of the three algorithms played each deal 75 times. The overall win rate for determinized UCT was 43.6%, for ISMCTS it was 42.3% and for cheating UCT it was 56.5%, which are approximately the same as those previously obtained for the original 1000 deals. This is unsurprising: the 1000 deals originally selected were chosen to

be a good indicator of typical playing strength. Each deal was then put into one of three categories according to the difference in win rate between cheating UCT and determinized UCT. If cheating UCT outperformed determinized UCT (with 95% significance) the deal was put into the category > 0. If determinized UCT outperformed cheating UCT (also with 95% significance) the deal was put into the category < 0. All other deals were put into the category ≈ 0. There are 1421 deals in category > 0, 3562 in category ≈ 0 and the remaining 17 in category < 0.

The win rates of ISMCTS and determinized UCT for the categories > 0 and ≈ 0 are shown in Figure 5.12. Since being in category < 0 is such a rare event, results for this category are not given. In the deals in which cheating UCT is significantly better than determinized UCT (category > 0), the win rate of ISMCTS is significantly better than that of determinized UCT. These deals are arguably those where knowing hidden information has an impact on the game and also deals where determinization may suffer from strategy fusion issues. In deals where there is no significant difference between cheating and determinization, it is observed that determinization is better than ISMCTS. It is arguable that hidden information has little impact on these deals, for example that one of the players has such a strong hand that a win is assured irrespective of what other players hold.

Despite the fact that overall the strength of ISMCTS and determinized UCT is approximately the same, there can be a great difference in the behaviour of the two depending on which cards are dealt. In the 1421 deals where a cheating UCT player outperforms determinized UCT, so does ISMCTS on average. This suggests that ISMCTS may be benefiting from a lack of strategy fusion issues along with the cheating player in these deals. It is an unusual feature of Dou Di Zhu among hidden information games that having access to hidden information only provides a strong advantage in a minority of deals, and has little effect in the rest.

In 3562 deals there is no significant difference between cheating UCT and determinized UCT. In these deals ISMCTS has a slightly lower win rate than determinized UCT. In these deals some factors other than hidden information and strategy fusion may be causing a detrimental effect on the performance of ISMCTS, but not on determinized UCT. The most significant difference between the two algorithms is the structure of the trees searched. The tree searched by ISMCTS offers several advantages over the determinization approach in general, but may be disadvantageous in certain deals. In the next section is it explained how this difference can be accounted for by an increase in branching factor in the ISMCTS algorithm, due to the nature of actions in Dou Di Zhu.

Influence of Branching Factor on Playing Strength The fact that there are some deals in which determinization outperforms cheating and many in which there is no difference between the two algorithms is a surprising feature of Dou Di Zhu, since intuitively the cheating player should have a strong advantage. One possible explanation for this is that branching factor has a large influence

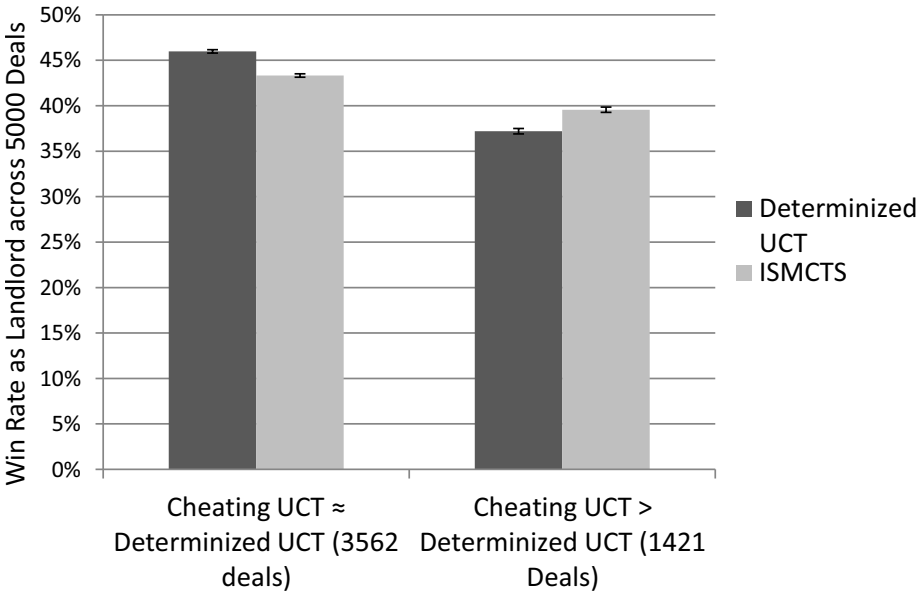

Figure 5.12: Playing strength of ISMCTS and Determinized UCT for Dou Di Zhu. The playing strength is shown for deals in which cheating UCT is better than determinized UCT (category > 0) or where the two algorithms perform approximately equally (category ≈ 0).

on the playing strength of these algorithms. In Dou Di Zhu, certain hands may have a large total number of moves available when making a leading play since there are many possible ways of choosing kicker cards to attach to a main group. Another feature of the game is that every move a player makes in the game is in the set of moves that player could make as a leading play from their starting hand. This set therefore forms an upper bound on the number of moves available in each state for a particular deal and if this set is large, there is likely to be many more nodes in the tree than if this set is small.

In the case that determinizations produce hands with completely different sets of moves, ISMCTS is at a disadvantage compared to determinized UCT. This is because ISMCTS will spend a lot of time adding new nodes near the root of the tree (since many determinizations will have unique moves that are not common to other determinizations) and consequently the statistics in the search tree will mostly be derived from random playouts near the root. On the other hand, determinizing players will be able to perform a deeper search for each determinization, since a large number of possible opponent moves will be ignored.

The following measurements were made for each of the 5000 deals tested:

- The total number of moves the non-Landlord players would be able to make as a leading play from their starting hand (using the actual cards these players hold for this particular deal).
- The average of the above for 200 random determinizations of the deal (where the cards held by the non-Landlord players are randomized).
- The average number of unique leading plays for non-Landlord players that are discovered in 40, 250 and 1000 determinizations, i.e. after generating a certain number of determinizations how many possible unique leading plays have been seen for the non-Landlord players.

These measurements, averaged across all 5000 deals, are presented in Table 5.3. It should be noted that these measurements are a function of the deal; the first measurement is exact for each deal, while the second depends on the sampled determinizations. These measurements were made only for the non-Landlord players since the playing strength experiments were conducted from the point of view of the Landlord player. This means the algorithms tested always had the same number of branches at nodes where the Landlord makes a move, since the Landlord can see his cards in hand. The first measurement is an indicator for the number of branches that may be expected at opponent nodes for the cheating UCT player as the Landlord. Similarly the second measurement indicates the number of branches for opponent nodes with determinized UCT as the Landlord. Both of these measurements are upper bounds, since if an opponent has played any cards at all then the number of leading plays will be smaller. The third, fourth and fifth measurements indicate how many expansions ISMCTS will be making at opponent nodes after a certain number of visits, since a new determinization is used on each iteration. Again this measurement

Measurement	**Result** (3 s.f.)
Total leading plays	87.7
Average leading plays per determinization	87.8
Average unique leading plays from 40 determinizations	754
Average unique leading plays from 250 determinizations	1230
Average unique leading plays from 1000 determinizations	1500

Table 5.3: Averages of different measurements of leading plays for the opponents in Dou Di Zhu across 5000 deals

is an upper bound since only one move is actually added per iteration and if there were moves unique to a determinization which were never seen again, only one of them would be added to the tree.

As seen in Table 5.3, from 1000 determinizations on average 1500 unique leading plays are seen and yet there are only approximately 88 unique leading plays for a particular determinization of a deal. What is apparent from these measurements is that there are a lot of moves available within the information set and only a small number of them are available within each state in the information set. After just 40 determinizations, ISMCTS will on average have seen nearly ten times as many unique moves as there are per determinization. This means that at nodes in the information set tree where an opponent makes a leading play, node expansion will happen for many more simulations than if the moves were derived from one single determinization. At other nodes in the tree where the opponent must play a certain move type, any move that either player could play will appear as branches at nodes for both opponents. This suggests that nodes in the tree corresponding to an opponent making a leading play act as a bottleneck for ISMCTS; the algorithm very rarely explores beyond these nodes with only 10000 simulations. With 250 simulations per determinization, it is likely that determinized UCT reaches a similar depth in the tree, which would explain why the overall performance of the two algorithms is broadly similar.

Another observation that can be made from these results is that the average number of leading moves for the actual state of the game and for each determinization is approximately the same. This is unsurprising since both measurements are derived from states constructed by randomly dealing unseen cards. This implies that cheating UCT and determinized UCT are searching trees of approximately the same size on average. Results from experiments earlier in this chapter suggest that the extra knowledge gained by cheating does not always provide a strong advantage in Dou Di Zhu.

Later in Chapter 8.2 it is shown that ISMCTS can be greatly improved for Dou Di Zhu by learning a simulation policy online (which would not be effective in PIMC due to the low number of MCTS iterations in each determinization). Dou Di Zhu also has many features which are unusual (and arguably pathological) for games of hidden information, for example quite often the precise cards

chosen as kickers are not of particular strategic importance; indeed attaching a card as a kicker is often a way of getting rid of a "useless" card that would be difficult to play otherwise yet the number of permutations of kicker choices contributes greatly to the branching factor of the game. In the next chapter (Figure 6.3, page 96) it is shown that the performance of ISMCTS is better than PIMC search in Lord of the Rings: The Confrontation, which lacks this pathology.

Playing strength against a commercial AI

To assess the absolute strength of PIMC search and ISMCTS for Dou Di Zhu, both were tested against a strong AI agent developed commercially by AI Factory Ltd [27] [2]. This agent uses flat Monte Carlo evaluation coupled with hand-designed heuristics. These results are presented in Figure 5.13. In addition the ISMCTS algorithm was more recently compared against a heuristic based AI developed by AI Factory Ltd for the game of Spades [10]. These results are presented in Figure 5.14.

For implementation reasons the methodology of these experiments differed from that of other experiments in this section. For Dou Di Zhu, the identity of the Landlord player is decided by a bidding phase according to standard game rules. All agents use the AI Factory agent's AI for bidding. Three algorithms are tested: the AI Factory agent, determinized UCT with 40 determinizations and 250 iterations per determinization, and ISMCTS with 10 000 iterations. Each plays 1000 games against two copies of the AI Factory agent. When all three agents are identical the expected number of wins for each is 500. A similar experiment was also performed for the AI in the AIFactory game Spades. Here ISMCTS was tested against the AI Factory AI at the normal and maximum levels where the normal level was identical to the hardest difficulty available in the product and the maximum level allows a very large amount of decision time. In each both Dou Di Zhu and Spades, both members of a partnership use the same AI (but do not "share" information).

Results of these experiments are shown in Figures 5.13 and 5.14, which indicate that both PIMC and ISMCTS significantly outperform the AI Factory players. In the case of Spades, ISMCTS is on par with the AI Factory AI even when it is allowed a very large computational budget. It should also be noted that in both games, the commercial AI includes many game specific heuristics, yet ISMCTS yields stronger playing strength with no knowledge. Thus it can be concluded in absolute terms that both PIMC and ISMCTS produce plausible, and indeed strong, play for Dou Di Zhu and Spades.

5.3 Summary

In this chapter the combination of PIMC and MCTS was investigated for several domains, primarily Dou Di Zhu and Lord of the Rings: The Confrontation. Ex-

[2] `www.aifactory.co.uk`.

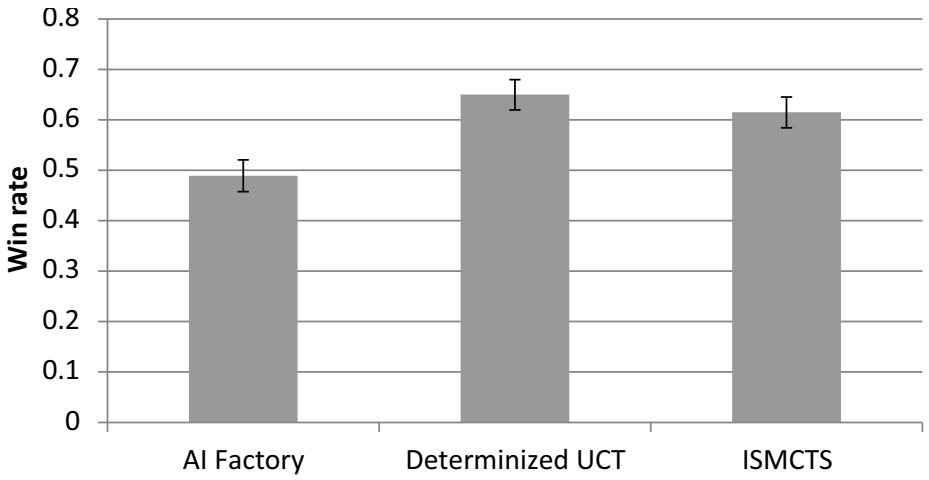

Figure 5.13: Playing strengths of ISMCTS and PIMC (Determinized UCT) against AI Factory AI for Dou Di Zhu.

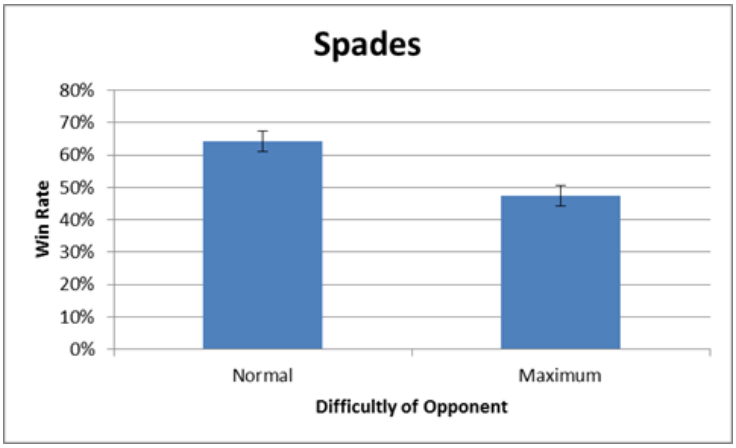

Figure 5.14: Playing strength of ISMCTS against AI Factory AI for spades at the normal and maximum levels

periments indicated that given a fixed computational budget, the performance of PIMC with MCTS is sensitive to the ratio between determinizations and MCTS iterations per determinization. Using more determinizations allows for a more diverse sample of possible states, but allows for less time to analyse each state given a fixed computational budget.

Experiments were performed in the game Mini Dou Di Zhu to quantify the effects of strategy fusion and non-locality (or lack of inference), two known issues with PIMC search. The results indicated that strategy fusion is a far more significant problem for Mini Dou Di Zhu, and the benefits of inference were comparatively small compared to overcoming strategy fusion. This lead to the introduction of the ISMCTS algorithm, which extends MCTS to searching trees of information sets and does not suffer from strategy fusion.

ISMCTS offers numerous advantages over PIMC search with MCTS in addition to removing the effect of strategy fusion. PIMC does not transfer knowledge between determinizations except in the final step of making a decision. ISMCTS constructs a single tree, so allowing information to be shared as the search progresses. ISMCTS also removes the need to find the correct balance between determinizations and simulations. The fact that other authors have investigated single tree approaches, combined with the wide range of good experimental results (including new results in this thesis) suggest that ISMCTS is a significant advance beyond PIMC.

When considering deals in which a cheating player outperforms a PIMC player significantly (implying that hidden information is important in that specific deal), it was shown that ISMCTS does outperform PIMC search. It is an unusual feature of Dou Di Zhu among hidden information games that having access to hidden information only provides a strong advantage in a minority of deals, and has little effect in the rest. Furthermore it was shown that Dou Di Zhu has several pathologies which lead to large branching factors at opponent nodes in ISMCTS trees, which do not affect PIMC search. When evaluating the overall performance of ISMCTS for Dou Di Zhu, ISMCTS failed to outperform PIMC search, however in the next chapter (Figure 6.3, page 96) it is shown that the performance of ISMCTS is better than PIMC search in Lord of the Rings: The Confrontation, which lacks this pathology.

Finally the playing strength of ISMCTS was established for the games Dou Di Zhu and Spades by comparing against benchmark commercial AI players developed by AI Factory Ltd (which are amongst the strongest available and contain extensive heuristics and optimizations). In Dou Di Zhu, both ISMCTS and PIMC search outperform the AI Factory AI. In the case of Spades, ISMCTS is on par with the AI Factory AI even when it is allowed a very large computational budget. Thus it can be concluded in absolute terms ISMCTS produces strong play for Dou Di Zhu and Spades.

Algorithm 2 Detailed psuedocode for the ISMCTS algorithm

function ISMCTS($[s_0]^{\sim_1}, n$)
  create a single-node tree with root v_0 corresponding to $[s_0]^{\sim_1}$
  for n iterations **do**
    choose $d_0 \in [s_0]^{\sim_1}$ uniformly at random
    $(v, d) \leftarrow$ SELECT(v_0, d_0)
    if $u(v, d) \neq \emptyset$ **then**
      $(v, d) \leftarrow$ EXPAND(v, d)
    $r \leftarrow$ SIMULATE(d)
    BACKPROPAGATE(r, v)
  return $a(c)$ where $c \in \arg\max_{c \in c(v_0)} n(c)$

function SELECT(v, d)
  while d is non-terminal and $u(v, d) = \emptyset$ **do**
    select[1] $c \in \arg\max_{c \in c(v,d)} \left(\frac{r(c)_{\rho(d)}}{n(c)} + k\sqrt{\frac{\log n'(c)}{n(c)}} \right)$
    $v \leftarrow c$; $d \leftarrow f(d, a(c))$
  return (v, d)

function EXPAND(v, d)
  choose a from $u(v, d)$ uniformly at random
  add a child w to v with $a(w) = a$
  $v \leftarrow w$; $d \leftarrow f(d, a)$
  return (v, d)

function SIMULATE(d)
  while d is non-terminal **do**
    choose a from $A(d)$ uniformly at random
    $d \leftarrow f(d, a)$
  return $\mu(d)$

function BACKPROPAGATE(r, v_l)
  for each node v from v_l to v_0 **do**
    increment $n(v)$ by 1
    $r(v) \leftarrow r(v) + r$
    let d_v be the determinization when v was visited
    for each sibling w of v compatible with d_v, including v itself **do**
      increment $n'(w)$ by 1

Algorithm 3 High-level pseudo-code for the Information Set MCTS (ISMCTS) algorithm.

```
1: function ISMCTS([s_0]^{~_1}, n)
2:     create a single-node tree with root v_0 corresponding to the root information set [s_0]^{~_1} (from player 1's viewpoint)
3:     for n iterations do
4:         choose a determinization d at random from [s_0]^{~_1}, and use only nodes/actions compatible with d this iteration
5:
6:         // Selection
7:         repeat
8:             descend the tree (restricted to nodes/actions compatible with d) using the chosen bandit algorithm
9:         until a node v is reached such that some action from v leads to a player 1 information set which is not currently
                 in the tree or until v is terminal
10:
11:        // Expansion
12:        if v is non-terminal then
13:            choose at random an action a from node v that is compatible with d and does not exist in the tree
14:            add a child node to v corresponding to the player 1 information set reached using action a and set it as the
                   new current node v
15:
16:        // Simulation
17:        run a simulation from v to the end of the game using determinization d
18:
19:        // Back-propagation
20:        for each node u visited during this iteration do
21:            update u's visit count and total simulation reward
22:            for each sibling w of u that was available for selection when u was selected, including u itself do
23:                update w's availability count
24:
25:    return an action from the root node v_0 such that the number of visits to the corresponding child node is maximal
```

Chapter 6

Partial Observability

This thesis investigates the application of Monte Carlo Tree Search to games of imperfect information. In particular, games with three different types of imperfect information:

- *Information sets* are collections of states, which appear in a game when one player has knowledge about the state that another player does not. For example, in a card game each player hides his own cards from his opponents. In this example the information set contains all states which correspond to all possible permutations of opponent cards. A player knows which information set they are in, but not which state within that information set.
- *Partially observable actions* appear in games where a player performs an action but some detail of the action is hidden from an opponent. In other words, the opponent observes that an action was taken from a set of possible actions, but not which action. Partially observable actions can also be taken by the environment player, for example to reveal a piece of information to a subset of the players.
- *Simultaneous moves* arise when multiple players reveal decisions simultaneously without knowing what decision the other players have made. The effect of the decisions is resolved simultaneously. The well known game of Rock-Paper-Scissors is an example of this. This can be considered as a special case of partially observable actions, where every player chooses an action secretly (in any order), then it is revealed which actions were chosen.

Chapter 5 introduced *Information Set Monte Carlo Tree Search (ISMCTS)*, that overcome some of the weaknesses of the *Perfect Information Monte Carlo* (PIMC) seach approach to handling hidden information. Instead of searching the minimax tree produced by a determinization, a tree where the nodes represent information sets rather than states is constructed. This offers the advantage that statistics about moves are collected together in one tree, thus using

the computational budget more efficiently, whereas PIMC search constructs several trees and does not share any information between them. Furthermore this approach offers an improved model of the decision making process compared to determinization, since the search is able to exploit moves that are good in many states in the information set and the effects of strategy fusion can be lessened or eliminated entirely.

This chapter covers work investigating how ISMCTS can be adapted to games with partially observable actions, where hidden information occurs in an action (rather than a state). Partial observability can be considered as just a special case of hidden information; indeed any information revealed or concealed by partially observable actions is represented in a player's information set. However for games with large number of information sets it is intractable to store all of them, which is why the ISMCTS algorithm groups together information sets which cannot be distinguished just by the action history of a game. The number of groups of information sets generated in this way is a manageable number, but introduces the problem of subset-armed bandits (discussed in Chapter 5.2.1, page 68). This process of information set grouping works well in practice for games where all actions are fully observable, however the grouping process breaks down when actions are partially observable because each player observes a different action history. In this chapter it is shown how this problem can be remedied by building multiple ISMCTS trees, which correctly model the observed action history for each player. This approach still has the benefit of not storing every information set, at the cost of greater memory usage and more groups of information sets compared to building a single ISMCTS tree. However the benefit of correctly handling partially observable actions outweighs the cost of building multiple trees. The experiments in this chapter focus on two games with partially observable actions:

- *Lord of the Rings: The Confrontation* [64] is a board game with elements similar to Stratego [97] and has several features which make it even more challenging from an AI perspective. It has hidden information, partially observable moves and simultaneous moves, all of which make the decision making process highly complex. The game also has an asymmetry between the two players since they have different win conditions and different resources available to them, which necessitates different tactics and strategies.

- m, n, k-*games* [98] are a generalization of games such as Noughts and Crosses and Renju where players try to place k pieces in a row on a $m \times n$ grid. In the *phantom* $4, 4, 4$-*game*, players cannot see each other's pieces and get feedback about the location of the other player's piece only when they attempt to play in an occupied square. This game is used as a test domain with hidden information and partially observable moves but low complexity.

Experiments in Section 5.2.4 showed that the relative performance of ISMCTS versus determinized UCT varies across these three domains. In Lord of the

Rings: The Confrontation a deep search is required for strong play, and so ISMCTS has the advantage due to its more efficient use of the computational budget. In the phantom $4,4,4$-game the effects of strategy fusion are particularly detrimental, so again ISMCTS outperforms determinization. This chapter introduces a new algorithm called *Multiple Observer ISMCTS* (MO-ISMCTS) which is designed to handle partially observable actions by building multiple ISMCTS trees.

6.1 ISMCTS and Partial Observability

In many games of imperfect information, pure policies can easily be exploited and thus a strong player must find a mixed policy. Rock-paper-scissors and Poker games are two examples where mixing strategies is important to achieving a strong level of play against players capable of recognising and exploiting pure policies. MCTS is not designed explicitly to seek mixed policies but often do so anyway, in the sense of choosing different actions when presented with the same state. This arises from the random nature of Monte Carlo simulation: the MCTS algorithm is not deterministic and even when pure policy optimal strategies exist MCTS is only guaranteed to find one of them. Shafiei et al [65] demonstrate that the UCT algorithm finds a mixed policy for rock-paper-scissors.

Experiments run during the development of the ISMCTS indicated that ISMCTS finds mixed ((non-equilibrium)) policies for the small, solved game of Kuhn poker [118]. However MCTS often fails to find optimal (Nash) policies for games of imperfect information. Ponsen et al [119] suggest that algorithms such as MCCFR [120] are a better fit for approximating Nash equilibria in games whose trees contain millions of nodes, whereas the strength of an MCTS approach lies in finding a strong suboptimal policy but finding it in reasonable CPU time for complex games with combinatorially large trees. More recently Heinrich and Silver demonstrate than the UCT algorithm can be modified to converge to a Nash-Equilibrium in Kuhn Poker [84].

The dealing of cards can be thought of as an action by an extra player (referred to as the environment) which always chooses actions randomly. Therefore the environment player chooses a random deal from all possible deals, but the players do not observe which deal was chosen by the environment (rather a set of possible deals consistent with the cards they were dealt). This is an example of a partially observable action, since all players observe that a deck ordering was chosen (taking an action), but not what the ordering was. This is handled in the ISMCTS algorithm essentially by considering all of these actions to be the same and constructing a single search tree for all choices of deck ordering that are consistent with the players observations of the game so far.

Applying the same method to partially observable actions taken by players during the game creates a new problem. Suppose that in a card game one player chooses a hidden card from their hand and places it face down on the table. The other players do not know the identity of this card but in an ISMCTS tree each different choice of card will lead to a unique branch, so in the sub-tree beneath

each branch it is implicitly assumed that the identity of this card is known to all players. ISMCTS makes the faulty assumption that the player can behave differently depending on the choice of actions made by other players when the player cannot observe which action was taken. In other words, partially observable actions lead to strategy fusion in ISMCTS. Experiments in Section 6.3.2 show that this effect is damaging to the performance of ISMCTS when partially observable moves occur, and the playing strength is weaker than the PIMC search algorithm.

To resolve this problem a similar approach could be taken to how strategy fusion was handled with ISMCTS, by merging together all branches where the action corresponds to the same observation made by the player. However this projects the issue onto the other players since now they cannot make different future decisions depending on the action which was chosen. In this chapter it is shown that the key to solving this problem is building a tree for each player, where the nodes in each tree correspond to information sets from the point of view of that player and branches correspond to the observation made by the player about the action which was chosen. This algorithm is called *multiple observer ISMCTS* (MO-ISMCTS) [27].

The advantage of the MO-ISMCTS algorithm over ISMCTS or PIMC search is that strategy fusion is eliminated since partially observable actions lead to a unique node in the search tree for the player making the action. Experiments in Section 6.3.2 demonstrate that MO-ISMCTS significantly outperforms other algorithms in the test domains. However the disadvantage is increased memory use (linearly with the number of players) and complexity of the algorithm, reducing the number of MCTS iteration that can be performed in a given time period relative to ISMCTS. However, experiments later in this chapter show that when a sufficiently large amount of CPU time is used for each decision, MO-ISMCTS outperforms ISMCTS (and PIMC). It should be noted that by creating multiple trees pre player, many of the nodes in each tree correspond to non-decision states for the owner of the tree. These extra nodes exist to facilitate the correct transition from one decision node to the next (depending on the observation of other players actions), but the statistics in these nodes are never used. It would be possible to reduce the memory footprint of the multiple tree algorithm significantly by removing these extra nodes, and instead implementing a mechanism to correctly manage the transitions between decision nodes. However it would be difficult to achieve this without introducing additional CPU overheads or changing the way in which the software framework represents games. Therefore the multiple tree representation was used for experiments in this, since it allows transitions between nodes to be managed efficiently in the software framework and high memory use was not a problem.

6.2 Bandit algorithms for Simultaneous Actions

This section summarizes how simultaneous actions were handled in ISMCTS in the various games used for experiments. There has been some existing work on

applying sampling based methods and bandit algorithms to simultaneous action selection which was surveyed in Chapter 3.2.4. This can be done by modelling the selection of actions by each player as independent multi-arm bandit problems, The reward distribution of each arm depends on the policies of the other players which may not be static, so this technique is not guaranteed to converge to any particular policy.

When choosing a sampling methods for use with ISMCTS, numerous sampling methods were investigated with mixed results, although the UCB1 algorithm appeared stronger than EXP3 in some cases. This matches the results of Perick et al [40], however the experiments in this Chapter used the EXP3 algorithm in an identical manner to Teytaud and Flory [42]. It is possible that better performance could be achieved by improving the bandit algorithm for simultaneous moves. Recent work by Lanctot [66] introduced the *Online Outcome Sampling* algorithm which has convergence guarantees and better empirical performance than the independent bandits algorithm. Integrating similar techniques with ISMCTS is a good subject for future work in this area.

6.3 ISMCTS with Partially Observable Moves

In this section the MO-ISMCTS algorithm is described in detail and the results of experiments are presented in Section 6.3.2. For clarity, ISMCTS is referred to as *Single Observer ISMCTS* (SO-ISMCTS) in discussion to distinguish it from MO-ISMCTS. Note that all variants of ISMCTS in this chapter are equivalent for a game with fully observable moves (and also equivalent to MCTS in perfect information games).

6.3.1 Algorithms

This section introduces two new algorithms which modify the ISMCTS algorithm to handle partially observable moves.

Single observer information set MCTS with partially observable moves (SO-ISMCTS+POM)

SO-ISMCTS does not completely avoid the problem of strategy fusion in games with partially observable moves, as all opponent moves are treated as fully observable. Suppose that the game in Figure 5.10 (page 74) is modified to include partially observable moves, so that player 2 cannot distinguish a_1 from a_2 nor a_3 from a_4 and player 1 cannot distinguish between b_1, b_2 and b_3. Here the search assumes that different actions can be taken in response to opponent actions b_1 and b_2, for instance, whereas in fact these actions are indistinguishable and lead to the same player 1 information set.

In SO-ISMCTS, edges correspond to actions, i.e. moves from the point of view of the player who plays them. In *single-observer information set MCTS with partially observable moves (SO-ISMCTS+POM)*, edges correspond to moves

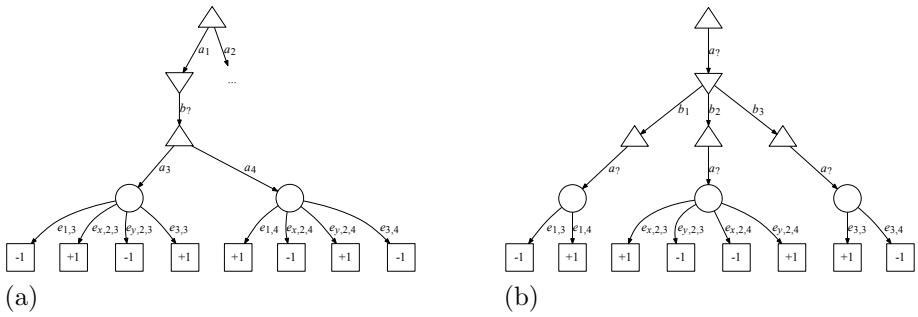

Figure 6.1: Information set search trees for the game shown in Figure 5.10 with partially observable moves, where player 2 cannot distinguish a_1 from a_2 nor a_3 from a_4 and player 1 cannot distinguish between b_1, b_2 and b_3. (a) shows the tree searched by SO-ISMCTS+POM; (a) and (b) together show the pair of trees searched by MO-ISMCTS, where (a) is from player 1's point of view and (b) from player 2's point of view.

from the point of view of the root player. Thus actions that are indistinguishable from the root player's point of view share a single edge in the tree. Figure 6.1 (a) shows the SO-ISMCTS+POM search tree for the game in Figure 5.10. The branches from player 1's decision nodes are unchanged; however, player 2's decision node now has a single branch corresponding to the single move from player 1's point of view, rather than one branch for each action.

As in SO-ISMCTS, each iteration is guided by a determinization. This raises the problem of how to update the determinization d according to a selected partially observable opponent move. For a determinization d and a move $[a]^{\smile 1}$, the set of compatible actions is $\alpha = A(d) \cap [a]^{\smile 1}$. If α is a singleton then its single element can simply be applied to d to obtain the determinization for the next level in the tree. If $|\alpha| > 1$ then an action from α is chosen uniformly at random, since the tree does not store any data with which to make a more informed choice. High level pseudocode for the SO-ISMCTS+POM is the same as Algorithm 3 for SO-ISMCTS (page 87) if "action" is replaced by "move (from player 1's viewpoint)".

Consider the example in Figure 6.1 (a). SO-ISMCTS+POM functions in much the same way as SO-ISMCTS (recall the example at the end of Section 5.2.3), except that branch $b_?$ is selected in place of b_2. When updating the determinization while descending the tree, an action must be applied corresponding to the selection of $b_?$. In this case, one of b_1, b_2 or b_3 is applied depending on which are legal actions in the current determinization. For each determinization, there are two possibilities, so one is chosen uniformly at random.

Multiple observer information set MCTS (MO-ISMCTS)

SO-ISMCTS+POM solves the strategy fusion problem of SO-ISMCTS (due to the presence of partially observable moves), at the expense of significantly weakening the opponent model: in particular, it is assumed that the opponent chooses randomly between actions that are indistinguishable to the root player. In the extreme case, when SO-ISMCTS+POM is applied to a phantom game (such as the phantom $4, 4, 4$-game) all opponent actions are indistinguishable and so the opponent model is essentially random.

To address this, *multiple observer information set MCTS (MO-ISMCTS)* is introduced. This algorithm maintains a separate tree for each player, whose nodes correspond to that player's information sets and whose edges correspond to moves from that player's point of view. Each iteration of the algorithm uses a specific determinization to descend all of the trees simultaneously. Each selection step uses statistics in the tree belonging to the player about to act in the current determinization to select an action. Each tree is then descended by following the branch corresponding to the move obtained when the corresponding player observes the selected action, adding new branches if necessary.

The information set trees can be seen as "projections" of the underlying game tree. Each iteration induces a path through the game tree, which projects onto a path through each information set tree. Figure 6.1 depicts these trees for the simple game of Figure 5.10: Figure 6.1 (a) corresponds to information sets and moves from player 1's point of view, and Figure 6.1 (b) from player 2's point of view.

The MO-ISMCTS approach is similar to the MMCTS algorithm proposed by Auger [101]. However there are several differences between MO-ISMCTS and MMCTS, the most important being that MO-ISMCTS uses determinizations to guide and restrict each search iteration whereas MMCTS does not. Also, whereas Auger [101] describes use of MMCTS in an offline manner (running the algorithm for a very large number of simulations and querying the resulting tree for decisions during play), MO-ISMCTS is designed for the more conventional online mode of play.

High level pseudocode for MO-ISMCTS is given in Algorithm 4 (page 109). Consider the example in Figure 6.1. First the algorithm generates a random determinization on each iteration. Then an action is selected from the root of Figure 6.1 (a) (i.e. player 1's tree), say a_1. The trees are descended by following the branch for the move corresponding to a_1, namely a_1 for player 1's tree and $a_?$ for player 2's tree. The state resulting from determinization d is updated by applying action a_1. Now have $\rho(d) = 2$ so Figure 6.1 (b) (player 2's tree) is used for selection, and an action legal in d is selected, say b_2. The trees are descended through $b_?$ and b_2 respectively, and d is updated by applying b_2. The selection process continues in this way. Backpropagation works similarly to SO-ISMCTS (as in the example at the end of Section 5.2.3, page 71), but updates all visited nodes (and their available siblings) in each player's tree.

Detailed Psuedocode

Detailed pseudocode for the MO-ISMCTS algorithm is presented in Algorithm 5 (page 110). This pseudocode uses the notation from Chapter 2, with the following additions:

- $a(v)$ = incoming move from player 1's point of view at node v
- $c(v, d) = \{u \in c(v) \ : \ m(u) \in M_1(d)\}$, the children of v compatible with determinization d
- $u(v, d) = \{m \in M_1(d) \ : \ \nexists c \in c(v, d) \text{ with } m(c) = m\}$, the moves from d for which v does not have children

The notation used for Algorithm 2 (see page 76) is also used with the following additional notations:

- v^i = a node in player i's tree
- $a(v^i)$ = incoming move from player i's point of view at node v^i

6.3.2 Experiments

In this section experiments are presented which compare the performance of SO-ISMCTS, PIMC and the new algorithms SO-ISMCTS+POM and MO-ISMCTS in the games Lord of the Rings: The Confrontation (LOTR:C) and the phantom $4, 4, 4$-game, both of which feature partially observable moves.

Experimental results for Lord of the Rings: The Confrontation

In this experiment, the following algorithms play in a round-robin tournament: cheating UCT, cheating PIMC search with UCT (referred to as dheating ensemble UCT), PIMC search with UCT (referred to as determinized UCT), SO-ISMCTS, SO-ISMCTS+POM and MO-ISMCTS. Each algorithm runs for 10 000 iterations per decision. Determinized UCT search uses 10 determinizations with 1 000 iterations for the Dark player, and applies all 10 000 iterations to a single determinization for the Light. These values were chosen based on the results presented in Section 5.1.1 (page 57). Cheating ensemble UCT uses 10 trees with 1 000 iterations each for both Light and Dark; devoting all iterations to a single tree would be equivalent to cheating single-tree UCT. The results of this experiment are shown in Figures 6.2 and 6.3.

Cheating single-tree UCT consistently outperforms the other algorithms by a large margin. For the Dark player, cheating ensemble UCT outperforms ISMCTS. However, for the Light player, cheating ensemble UCT and ISMCTS are on a par. This is slightly surprising, and would seem to suggest that the benefit of cheating is balanced by the increased depth to which ISMCTS is able to explore the tree (due to devoting all of its iterations to a single tree). That

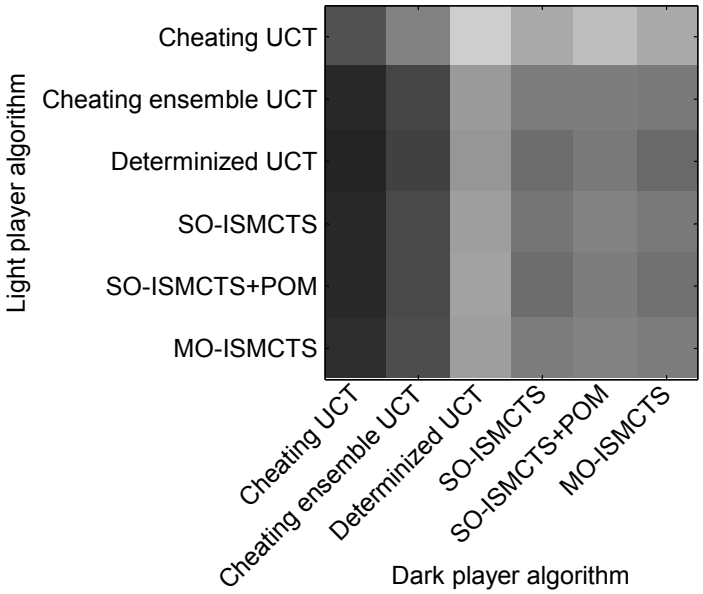

Figure 6.2: Heat map showing the results of the LOTR:C playing strength experiment. A white square would indicate a 100% win rate for the specified Light player algorithm against the specified Dark player algorithm, while a black square would indicate a 100% win rate for Dark against Light. Shades of grey interpolate between these two extremes.

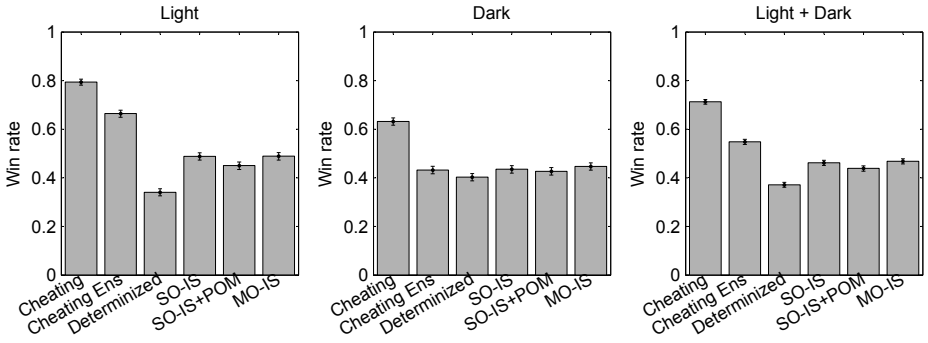

Figure 6.3: Results of the playing strength experiment for LOTR:C. In the "Light" graph, each algorithm indicated on the x-axis plays an equal number of games as the Light player against each algorithm as the Dark player, and the proportion of wins averaged over all Dark algorithms is plotted. The "Dark" graph is similar, with Light and Dark players interchanged. The "Light + Dark" graph averages these results for each algorithm regardless of player identity. In all cases, error bars show 95% confidence intervals.

this only holds true for one of the players may shed some light on the differences in approaches required for strong Light and Dark play in LOTR:C: as discussed in Section 5.1.1, to Dark the locations of Light's characters are the most important factor, but to Light it is equally important to be able to plan further ahead.

For the Dark player, PIMC search with UCT is outperformed by the other algorithms by a large margin. In particular, PIMC search with UCT is outperformed by all three ISMCTS variants. The success of ISMCTS here is probably due to the reduction in the effects of strategy fusion caused by using a tree of information sets, as well as the additional tree depth that arises by collecting all simulations in a single tree. For the Light player PIMC search with UCT is also worse than ISMCTS, but less dramatically so: the difference between PIMC search with UCT and MO-ISMCTS is around 4.4%, which is significant with 95% confidence. Since the Light player devotes all its computational resources to a single determinization the tree depth argument does not hold, but evidently there is still some benefit to the ISMCTS approach over PIMC search with UCT, most likely the ability to consider multiple determinizations without the impact of strategy fusion.

There is no significant difference in playing strength between the variants of ISMCTS. SO-ISMCTS+POM seems to perform slightly worse than the other variants of ISMCTS, but this difference is not statistically significant. That there is no significant difference between the algorithms seems to imply that the strategy fusion effects of assuming that opponent moves are fully observable in SO-ISMCTS, and the assumption that the opponent values indistinguishable actions equally in SO-ISMCTS+POM, are not as harmful as intuition may suggest.

As noted above, each trial in this experiment starts from the same hand-designed initial setup. The experiment was repeated with each game beginning from a different randomly generated initial setup. This biases the game slightly towards the Dark player, since a random initial setup is more likely to disadvantage the Light player (e.g. by placing Frodo in a vulnerable starting position). The same number of trials was carried out (750 for each combination of players) in order to achieve similar confidence intervals. Similar results to those above were observed, which support the same conclusions.

These experiments assess the relative playing strengths of several algorithms for LOTR:C, but gives no indication of their absolute strength. There is currently no known existing AI, commercial or otherwise, for this game. To test the playing strength of MO-ISMCTS, several games were played between an MO-ISMCTS agent and a range of human opponents. The human opponents can be characterised as experienced game players with two having significant experience with LOTR:C and five less experienced players.

For this experiment, playing all games with the same initial setup would not be a fair test: the AI agents cannot learn the opponent's initial setup between games, but a human player certainly can. Random initial setups were used, with constraints to avoid generating particularly bad placements: three strategically important characters (Frodo, Sam and the Balrog) are always placed in their

player's home cell. Since this information is known to the human player, it is also made available to the AI agent by appropriate construction of the initial information set.

When humans play LOTR:C, the game is partly a test of memory: one must remember the identities of revealed enemy characters. Since this is trivial for the AI agent, the graphical interface makes this information available to the human player. This ensures that the human and AI players are compared solely on the strength of their decisions, and not on the inherent advantage of a computer player in a test of memory.

32 games were played with a human player as Dark and the MO-ISMCTS player as Light, and 32 games with a human as Light and MO-ISMCTS as Dark. MO-ISMCTS achieved 14 wins as Light and 16 as Dark. MO-ISMCTS was evenly matched with intermediate to expert human players, so it may be concluded that MO-ISMCTS achieved strong play in an absolute sense (if the goal is to create an AI that is sufficiently challenging for humans to play against without any domain knowledge).

The players observed anecdotally that MO-ISMCTS plays highly plausible moves, and is particularly adept at engineering favourable endgame scenarios. Its weakest aspect is card play during combat: for example, it has a tendency to waste its more powerful cards in situations where less powerful cards would suffice. Presumably this occurs when the agent does not search deeply enough to see the value of holding onto a more powerful card until later in the game.

Experimental results for the phantom $4, 4, 4$-game

In this experiment, the six algorithms listed in Section 6.3.2 that were applied to LOTR:C again play a round-robin tournament. Each algorithm uses a total of 10 000 iterations, with cheating ensemble UCT and determinized UCT using 40 trees with 250 iterations per tree.

Results of this experiment are shown in Figures 6.4 and 6.5. From the "Players 1 + 2" graph in Figure 6.5 (c) the algorithms can be ordered from best to worst as follows, with statistical significance at 95% confidence in each case:

1. Cheating ensemble UCT;
2. Cheating UCT;
3. MO-ISMCTS;
4. Determinized UCT;
5. SO-ISMCTS;
6. SO-ISMCTS+POM.

Unsurprisingly, the cheating players perform best. The determinization approach appears to be strong for this game, although not as strong as MO-ISMCTS.

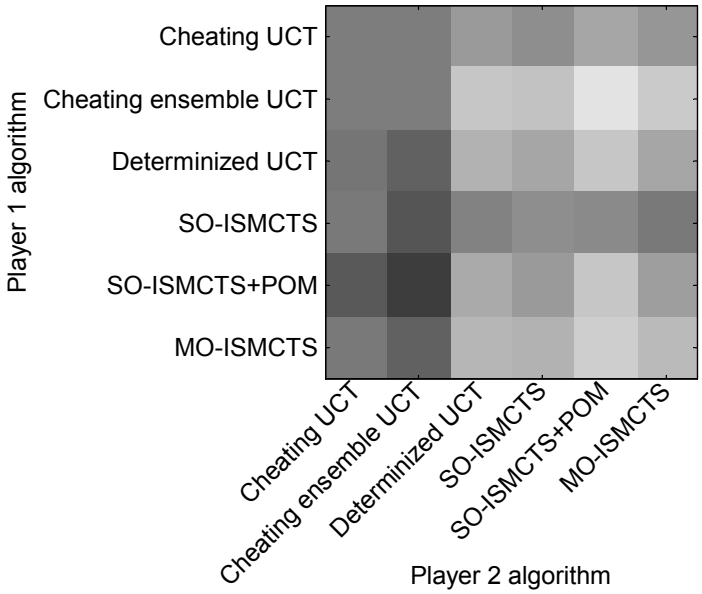

Figure 6.4: Heat map showing the results of the phantom $4, 4, 4$-game playing strength experiment. A white square would indicate a 100% win rate for the specified player 1 algorithm against the specified player 2 algorithm, while a black square would indicate a 100% win rate for player 2 against player 1. Shades of grey interpolate between these two extremes.

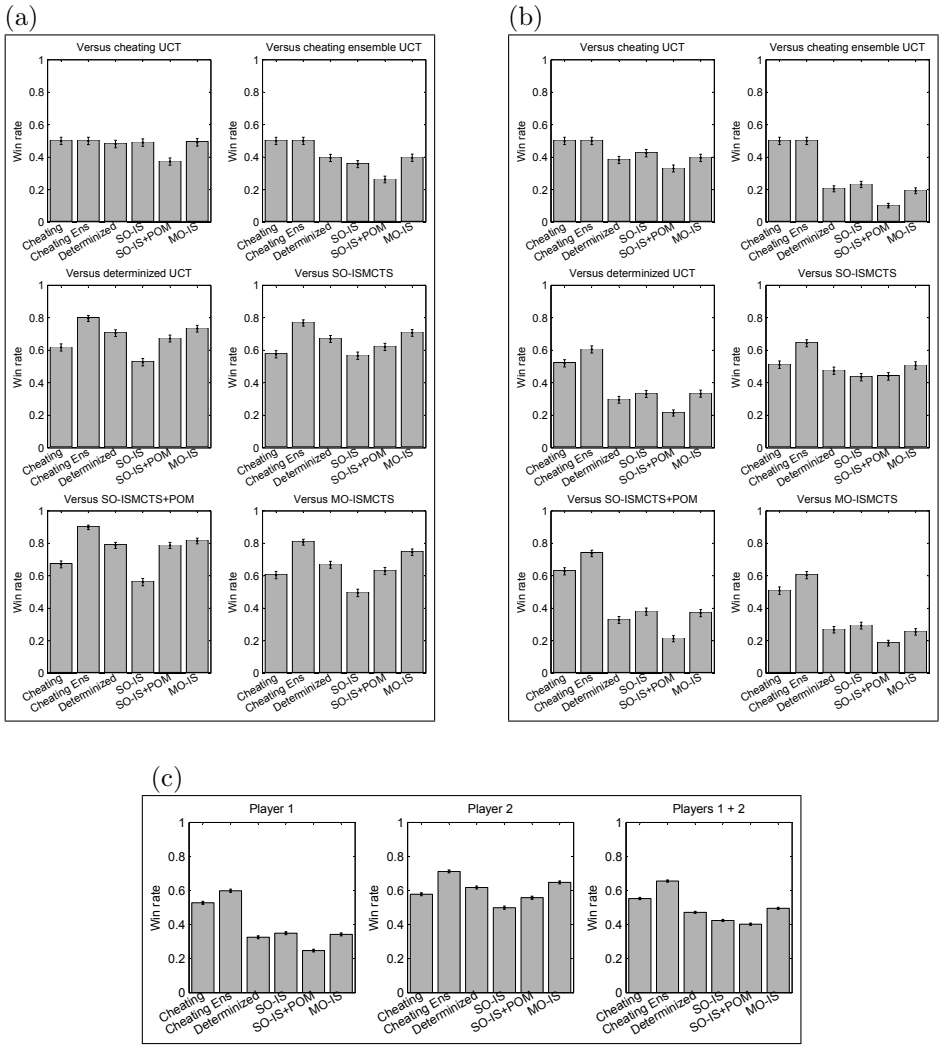

Figure 6.5: Results of the playing strength experiment for the phantom $4, 4, 4$-game. In (a), each graph compares the win rates of various player 1 algorithms against the player 2 opponent algorithm specified above the graph. In (b), each graph compares player 2 algorithms against a player 1 opponent. In (c), these results are aggregated into single graphs, each bar giving the win rate for its algorithm averaged over all tested opponent algorithms. In all cases, error bars show 95% confidence intervals.

There is some asymmetry between the two players in terms of the relative strengths of the algorithms. For player 1, SO-ISMCTS and MO-ISMCTS are on a par while SO-ISMCTS+POM underperforms. For player 2, SO-ISMCTS is outperformed by SO-ISMCTS+POM which is in turn outperformed by MO-ISMCTS. The three algorithms differ mainly in the assumptions they make about future play. SO-ISMCTS assumes that all actions are fully observable, which is both optimistic (I can respond optimally to my opponent's actions) and pessimistic (my opponent can respond optimally to my actions). SO-ISMCTS hence suffers from strategy fusion, since it is assumed the agent can act differently depending on information it cannot observe. In a phantom game, SO-ISMCTS+POM optimistically assumes that the opponent plays randomly. MO-ISMCTS's opponent model is more realistic: each opponent action has its own statistics in the opponent tree and so the decision process is properly modelled, but whichever action is selected leads to the same node in the player's own tree thus preventing the player from tailoring its response to the selected action. This addresses the strategy fusion problem which affects SO-ISMCTS.

Since player 1 has the advantage of moving first, it seems likely that these optimistic and pessimistic assumptions will have varying degrees of benefit and detriment to the two players. For example, a pessimistic player 2 algorithm may conclude (incorrectly) that the game is a loss, and so make poor decisions from that point. In short, it can be argued that solving the problem of strategy fusion is the key to strong play in the phantom $4, 4, 4$-game. Of the three ISMCTS variants, MO-ISMCTS is the most successful in overcoming strategy fusion. Indeed, the two SO-ISMCTS variants suffer more from the effects of strategy fusion than does determinized UCT.

One weakness of a cheating player is that it is overly pessimistic regarding the strength of its opponent. In particular, it assumes the opponent also cheats. In the phantom $4, 4, 4$-game, it often arises that the current state is a draw in the perfect information game but the non-cheating player has insufficient information reliably to force the draw. In other words, there are states where a non-cheating opponent is likely to choose an action that a cheating player would consider a mistake. If the cheating player could direct the game towards these states it would often win, but it sees no incentive to aim for these states in preference to any other state that leads to a draw. A non-cheating player rarely suffers from this problem, as it generally lacks the information to identify the state as a draw in the first place. It should be noted that this never causes a cheating player to lose a game, only to draw a game that it could conceivably have won. For this experiment the cheating algorithms played a total of 37 880 games, and did not lose a single game.

The above is a possible explanation for why cheating ensemble UCT outperforms cheating single-tree UCT. The former searches less deeply, and so its estimates for the game-theoretic values of states are less accurate. When the values of states are influenced more by random simulations than by the tree policy, there is a natural tendency to overestimate the value of states in which the opponent has more opportunities to make a mistake. Similar observations discussed in Section 5.1.2 were made on the propensity of non-cheating players

to make mistakes, and the benefit to a cheating minimax player of a tie-breaking mechanism that favours states from which the opponent has more suboptimal moves available.

Computation Time

It has been demonstrated that SO-ISMCTS and MO-ISMCTS offer advantages over PIMC search with UCT, however both of these algorithms are more complex and computationally expensive. Experiments so far have performed a fixed number of iterations without consideration of the algorithm used. It could be that a simpler algorithm could perform more iterations in the same amount of time as a more complex algorithm and achieve a better result. This sort of comparison is dependent on the efficiency of the implementation of each algorithm and may be difficult to test in practice. Instead, it has been observed that MCTS algorithms can reach a point where additional simulations leads to diminishing returns in terms of playing strength. If two MCTS based algorithms reach this point (independent of the efficiency of implementations) and are using the same amount of time to make decisions, then their relative strengths should not change much as more time is used. In this section it is demonstrated that with enough time per decision, the results obtained lead to the same conclusions as in previous experiments.

First an experiment was conducted where determinized UCT, SO-ISMCTS and MO-ISMCTS made the first decision for games of Dou Di Zhu, LOTR:C and the phantom $(4, 4, 4)$ game. Each algorithm used 10 000 simulations (with 40 trees and 250 iterations per tree for determinized UCT) and the average time to make a decision was recorded from 25 trials for each game. This was performed on a desktop PC running Windows 7 with 6GB of RAM and a 2.53GHz Intel Xeon E5630 processor. These results were used to calculate the number of iterations each algorithm could perform in 1 second for each game. The results are presented in Figure 6.6.

It is clear from Figure 6.6 that SO-ISMCTS and MO-ISMCTS are 2 to 4 times slower than determinized UCT and also that the game being played has an impact on the amount of time it takes to perform an MCTS iteration. In order to compare algorithms based on the amount of decision time, it was important to remove factors which affect the execution time of the experiments: experiments were run on a cluster of heterogeneous machines, all of which have other processes running at the same time. The algorithms were tested with a fixed number of iterations corresponding to a certain amount of decision time, assuming the rate of iterations per second for each algorithm/game in Figure 6.6. Approaches that build larger trees have increasing overheads per iteration, for example due to MCTS selection being applied to more nodes in the tree. It is reasonable to assume the rate of iterations per second from Figure 6.6, since after a few hundred iterations the overheads increase slowly.

For the phantom $4, 4, 4$-game, the three algorithms already take less than a second to execute 10 000 MCTS iterations due to the simplicity of the game logic. However for Dou Di Zhu and LOTR:C, it can be seen that in 1 second of

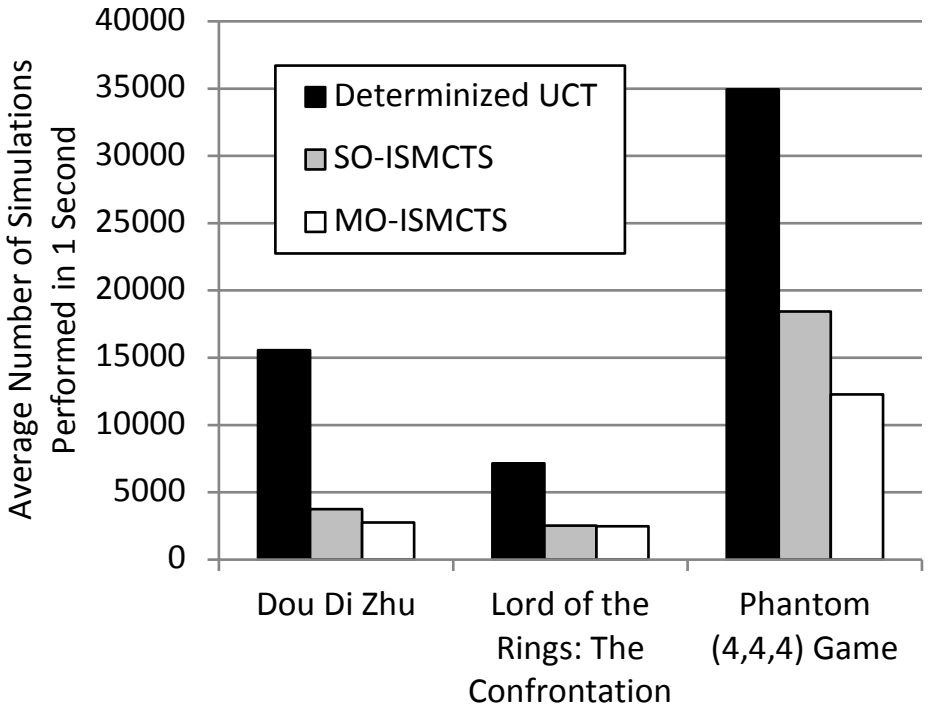

Figure 6.6: Average number of simulations performed in 1 second by determinized UCT, SO-ISMCTS and MO-ISMCTS on a desktop PC running Windows 7 with 6GB of RAM and a 2.53GHz Intel Xeon E5630 processor

decision time SO-ISMCTS and MO-ISMCTS execute around a third the number of iterations that determinized UCT does. From Figure 6.6 is it clear that the MO-ISMCTS implementation has some additional overheads, since it performed fewer iterations per second for Dou Di Zhu than SO-ISMCTS, although the two algorithms are equivalent for Dou Di Zhu, since it has no partially observable moves.

The performance of these algorithms when decision time is a factor was investigated for all three games. For Dou Di Zhu, PIMC search with UCT was compared to SO-ISMCTS with 0.25–8 seconds of decision time. In each case the algorithms played as the Landlord, with the non-Landlord players using PIMC search with UCT with 40 determinizations and 250 iterations per determinization. Playing strength was measured across the 1000 deals chosen for previous experiments in Dou Di Zhu. The number of trees and iterations for PIMC search with UCT was chosen for each total number of iterations to preserve the ratio of trees to iterations as 40/250. These results are shown in Figure 6.7. The relative playing strength of each algorithm was not significantly different to the results obtained in Section 5.2.4 for any amount of decision time (although SO-ISMCTS appeared slightly weaker with less than 1 second of decision time). This supports the conclusion from Section 5.2.4 that after reaching a certain depth, SO-ISMCTS spends many simulations expanding opponent decision nodes near the root of the tree and does not improve in playing strength.

For LOTR:C, MO-ISMCTS was compared to PIMC search with UCT for 1–30 seconds of decision time where PIMC search with UCT used 1 tree when playing as the Light player and a ratio of trees to iterations of 10/1000 when playing as the Dark player (these values were optimised in Section 5.1.1. The two algorithms played each other as both the Dark player and the Light player 500 times. The results are presented in Figure 6.8. For 1 second of decision time, MO-ISMCTS is slightly inferior to determinized UCT, but when at least 3 seconds of decision time is used MO-ISMCTS is significantly stronger than determinized UCT. The results in Figure 6.8 indicate that with a sufficient amount of decision time MO-ISMCTS offers a clear advantage over determinized UCT, with the difference between the two algorithms diverging given more CPU time.

For the phantom $4, 4, 4$-game, PIMC search with UCT, SO-ISMCTS and MO-ISMCTS were compared for 0.25–5 seconds of decision time per move. For each pair of algorithms, 500 games were played with each algorithm playing as player 1 and as player 2. The results are presented in Figure 6.9. When the CPU time used is less than 1.5 seconds per move the results are not significantly different to those for 10000 iterations presented in Section 6.3.2, with MO-ISMCTS slightly stronger than determinized UCT and clearly stronger than SO-ISMCTS. There is also a clear advantage of going first over going second. Above 1.5 seconds per move, the MO-ISMCTS algorithm continues to outperform SO-ISMCTS, in terms of results of games between these algorithms and performance against determinized UCT. However, both of these algorithms become relatively weaker than determinized UCT with increasing CPU time.

PIMC search with UCT implicitly assumes perfect information for both play-

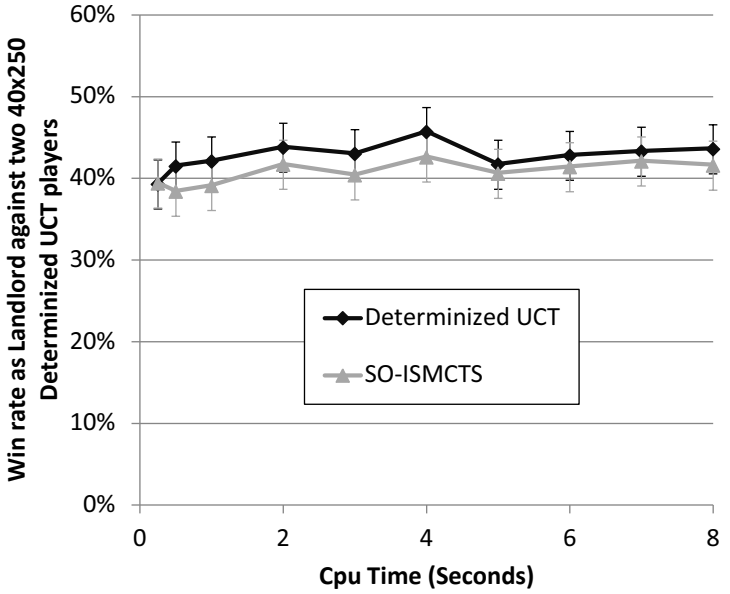

Figure 6.7: Playing strength of determinized UCT and SO-ISMCTS for different amounts of decision time playing Dou Di Zhu

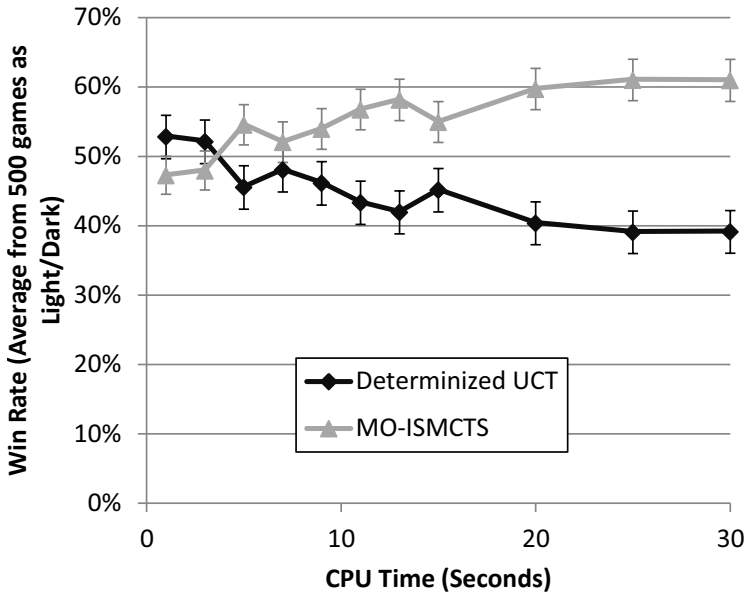

Figure 6.8: Playing strength of determinized UCT and MO-ISMCTS for different amounts of decision time playing LOTR:C

ers. The SO-ISMCTS and MO-ISMCTS players do not assume knowledge of hidden information. However, SO-ISMCTS does make the pessimistic assumption that the opponent has perfect information. MO-ISMCTS improves on this, supposing that the opponent knows the root state but not the moves made by the MO-ISMCTS player. Two properties of the phantom $4, 4, 4$-game are important here: the game is a loss if the opponent observes the game state at a crucial moment, even if he does not cheat subsequently; and the game is simple enough that MO-ISMCTS with more than 1.5 seconds can search a significant proportion of the entire game tree. The pessimism of the assumption that the opponent knows some or all of the hidden information often leads SO-ISMCTS and MO-ISMCTS to conclude, incorrectly, that the game is a loss, and thus play randomly since all lines of play have the same reward value. PIMC search with UCT has the more balanced, although highly inaccurate, view that both players can see all hidden information. This is consistent with the observations made from the fixed iteration experiments in the phantom $4, 4, 4$-game presented in Section 6.3.2.

Dou Di Zhu and LOTR:C are more complex than phantom $4, 4, 4$ so that it is not practical to search a substantial fraction of the whole tree within a reasonable time. Furthermore both Dou Di Zhu and LOTR:C remain difficult to win even when hidden information is known. Hence the reduction in playing strength for SO-ISMCTS and MO-ISMCTS with increasing CPU time is not seen. This is an inherent flaw in ISMCTS and prevents the algorithm from converging to optimal policies. Chapter 7 shows how this limitation can be overcome, by searching determinizations of every players information sets. In domains small enough for this effect to be problematic, it is expected that MC-CFR [120, 121] would produce better results given the convergence guarantees.

6.4 Summary

In this chapter variants of ISMCTS were introduced to improve the performance of the algorithm relative to PIMC search in domains with partially observable actions. In particular, the MO-ISMCTS algorithm searches multiple trees, where each tree is associated with a player. The SO-ISMCTS algorithm addresses the issue of wrongly assuming the player can distinguish between two states in an information set. The MO-ISMCTS algorithm additionally addresses the issue of wrongly assuming the player can distinguish between different partially observable moves made by an opponent.

MO-ISMCTS was investigated in two experimental domains: a complex board game (Lord of the Rings: The Confrontation) and a simple phantom game (the phantom $4, 4, 4$-game). In Lord of the Rings: The Confrontation it was shown that ISMCTS significantly outperforms PIMC search with UCT. Additionally it was shown that using a fixed CPU budget, the performance strength of MO-ISMCTS against PIMC search increased with additional CPU time. There was no significant difference in playing strength between the three variants of ISMCTS for LOTR:C. This seems to suggest that strategy fusion is

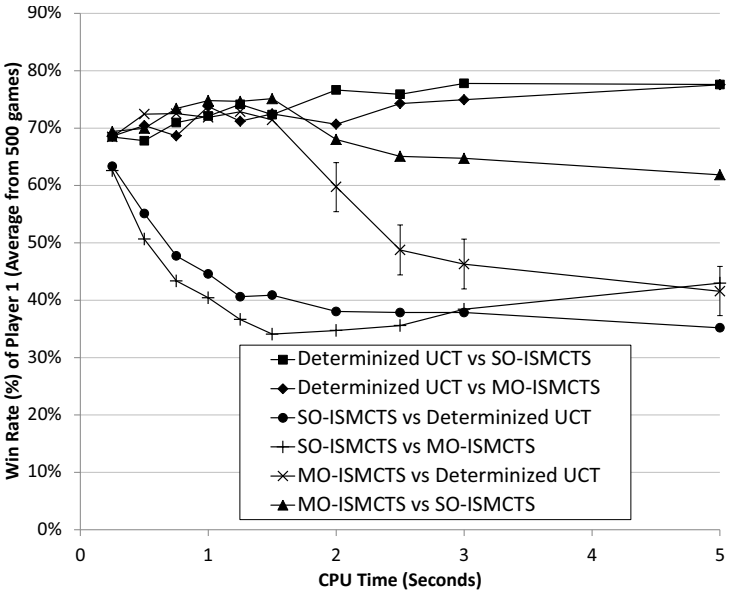

Figure 6.9: Playing strength of determinized UCT and SO/MO-ISMCTS for different amounts of decision time playing the phantom $4, 4, 4$-game. For each pair of algorithms A vs B, win rates are presented for algorithm A where algorithm A makes the first move. Error bars show 95% confidence intervals for MO-ISMCTS versus determinized UCT for ≥ 2 seconds per decision, and are representative of the size of error bars for the other series.

not a major factor in this game: the assumption that the identities of opponent characters are revealed when they move (i.e. that actions are fully observable) appears not to be exploited, or if it is exploited then this is not ultimately detrimental.

However, in the phantom $4, 4, 4$-game MO-ISMCTS significantly outperforms the other ISMCTS variants. This is unsurprising: in a phantom game, SO-ISMCTS suffers from strategy fusion, and SO-ISMCTS+POM assumes random opponent play. The gap between determinized UCT and MO-ISMCTS is smaller than for LOTR:C, and indeed the SO-ISMCTS variants fail to outperform PIMC search with UCT. Compared to LOTR:C and Dou Di Zhu, the phantom $4, 4, 4$-game tree is relatively small, and the game is more tactical than strategic, so issues such as search depth do not have such a dramatic effect on playing strength.

It was shown that one limitation of MO-ISMCTS is that the opponents have access to the player's hidden information: when the player chooses a determinization to use during the search, it does not determinize its own cards or the locations of its own pieces. Essentially the searching player assumes a cheating opponent, which is a worst case assumption but does mean that the opponent's lack of information can never be exploited. Furthermore, the assumption will be particularly harmful in games where there are no strategies that offer a chance of winning against a cheating opponent. This problem was observed in Section 6.3.2 when larger amounts of CPU time were used for MO-ISMCTS playing the phantom $4, 4, 4$-game. However, the solution is not as simple as merely randomising one's own information during determinization, as this destroys the player's ability to plan ahead beyond its current move (the searching player then assumes that it will forget its own information after the current turn). Addressing this issue, and particularly striking a balance between considering the actual situation and considering the other situations that the opponent thinks are possible, is important in games where information hiding is a significant part of successful play. This problem is addressed in Chapter 7.

Algorithm 4 High-level pseudocode for the multi-observer information set MCTS (MO-ISMCTS) algorithm.

1: **function** MO-ISMCTS($[s_0]^{\sim_1}, n$)
2: for each player $i = 0, \ldots, \kappa$, create a single-node tree with root v_0^i (representing $[s_0]^{\sim_1}$ from player i's viewpoint)
3: **for** n iterations **do**
4: choose a determinization d at random from $[s_0]^{\sim_1}$, and use only nodes/actions compatible with d this iteration
5:
6: *// Selection*
7: **repeat**
8: descend all trees in parallel using a bandit algorithm on player ρ's tree whenever player ρ is about to move
9: **until** nodes $v^0, \ldots, v^\rho, \ldots, v^\kappa$ are reached in trees $0, \ldots, \kappa$ respectively, player ρ is about to move at node v^ρ, and
some action from v^ρ leads to a player ρ information set which is not currently in the player ρ tree **or until** v^ρ
is terminal
10:
11: *// Expansion*
12: **if** v^ρ is nonterminal **then**
13: choose at random an action a from node v^ρ that is compatible with d and does not exist in the player ρ tree
14: **for** each player $i = 0, \ldots, \kappa$ **do**
15: if there is no node in player i's tree corresponding to action a at node v^i, then add such a node
16:
17: *// Simulation*
18: run a simulation from v^ρ to the end of the game using determinization d, (starting with action a if v^ρ is nonterminal)
19:
20: *// Backpropagation*
21: **for** each node u^i visited during this iteration, for all players i **do**
22: update u^i's visit count and total simulation reward
23: **for** each sibling w^i of u^i that was available for selection when u^i was selected, including u^i itself **do**
24: update w^i's availability count
25:
26: **return** an action from $A([s_0]^{\sim_1})$ such that the number of visits to the corresponding child of v_0^1 is maximal

Algorithm 5 Detailed pseudocode for the MO-ISMCTS algorithm

function MO-ISMCTS($[s_0]^{\sim_1}, n$)
  for each player i **do**
    create a single-node tree with root v_0^i
  for n iterations **do**
    choose $d_0 \in [s_0]^{\sim_1}$ uniformly at random
    $(v^0, \ldots, v^\kappa, d) \leftarrow$ SELECT$(v_0^0, \ldots, v_0^\kappa, d_0)$
    if $u(v^{\rho(d)}, d) \neq \emptyset$ **then**
      $(v^0, \ldots, v^\kappa, d) \leftarrow$ EXPAND$(v^0, \ldots, v^\kappa, d)$
    $r \leftarrow$ SIMULATE(d)
    for each player i **do**
      BACKPROPAGATE(r, v^i)
  return $a(c)$ where $c \in \arg\max_{c \in c(v_0^1)} n(c)$

function SELECT$(v^0, \ldots, v^\kappa, d)$
  while d is nonterminal and $u(v^{\rho(d)}, d) = \emptyset$ **do**
    select^0 $c \in \arg\max_{c \in c(v^{\rho(d)}, d)} \left(\frac{r(c)_{\rho(d)}}{n(c)} + k\sqrt{\frac{\log n'(c)}{n(c)}} \right)$
    for each player i **do**
      $v^i \leftarrow$ FINDORCREATECHILD$(v^i, a(w))$
    $d \leftarrow f(d, a(c))$
  return $(v^0, \ldots, v^\kappa, d)$

function EXPAND$(v^0, \ldots, v^\kappa, d)$
  choose a from $u(v^{\rho(d)}, d)$ uniformly at random
  for each player i **do**
    $v^i \leftarrow$ FINDORCREATECHILD(v^i, a)
  $d \leftarrow f(d, a)$
  return $(v^0, \ldots, v^\kappa, d)$

function FINDORCREATECHILD(v^i, a)
  if $\exists c \in c(v^i)$ with $a(c) = a / \backsim_i$ **then**
    return such a c
  else
    create and **return** such a c

Chapter 7

Bluffing and Inference

An important strategic aspect of hidden information games is the gathering and hiding of information. In many games players can gather extra information by extrapolating the likely information sets of other players based on actions they have chosen in a process called *Inference*. For example in a card game, if a player chooses not to play a winning card it is reasonable to assume they do not hold it in their hand. However if players are making reasonable inferences, this can be exploited by making moves that suggest a different information set to a players actual information set in a technique called *Bluffing*. For example in a poker game a player can make a large bet suggesting a strong hand when their hand is actually weak.

This chapter investigates how bluffing and inference can be achieved with the ISMCTS algorithm. Inference does not happen in the ISMCTS algorithm since the observed actions by other players do not influence the likelihood of a particular determinization being chosen. Therefore ISMCTS suffers from the non-locality problem with determinization, where states which are impossible if players are behaving rationally are considered possible. Furthermore the ISMCTS algorithm cannot exploit knowledge of another players information set which arises from their play. This is important in the game *The Resistance* (introduced in Chapter 4.2.11, page 47) which is studied in this chapter, where players must deduce the identity of other players from the decisions they make.

Bluffing is impossible with the basic ISMCTS algorithm (both multi and single observer), since all determinizations are from the searching players own information set. This leads to the assumption that other players have perfect information whilst the searching player does not. In many games this is a worst case scenario (that other players have perfect inference) and ISMCTS can still produce strong play when the opponents have limited opportunity to exploit knowledge of hidden information. For example in card games such as Dou Di Zhu and Spades where ISMCTS produces strong play, it is often the case that strategies can be constructed which are robust to the choices of other players (for example playing a long series of boss cards in Spades guarantees each trick will be won). However in other games if players are assumed to have

perfect inference, then there may be no winning strategies. This was seen for the Phantom $(4, 4, 4)$ game, where the performance of MO-ISMCTS degrades if there are enough iterations to allow the other player to exploit perfect inference.

Algorithms which approximate a Nash-equilibrium are capable of performing bluffing and inference, however a game theoretic approach to bluffing and inference requires the states of a game to be enumerable (in order to compute a policy at every information set) and for convergence guarantees, each unique information set to have a unique node in a search tree. This is intractable for many interesting hidden information games since the number of information sets is very large. One solution to this is to reduce a game to a smaller sub-game which is solvable then applying the sub-game strategy to the larger game in a process known as abstraction. This approach has a great deal of success in domains such as Poker [80], but requires a sub-game reduction to be designed for each game.

One of the most useful properties of MCTS is generality and applicability to large problems without needing significant domain knowledge, therefore the issue of finding suitable sub-game reductions is not addressed in this work. The main game studied is this chapter is The Resistance, which is small enough that using a unique node for each information set is tractable with ISMCTS. This representation is necessary in The Resistance to account for the fact that the value of an action depends heavily on the hidden information (as in a move that is good for Spy is bad for a member of the Resistance) in contrast to games where the value of an action is correlated across information sets (as in the opponent will play the winning card if they hold it and avoid the losing card whether or not they hold it). Details of how this one node per information set representation is generated are presented in Section 7.2.

It has been proven by Silver et al [82] that algorithms like ISMCTS will propagate the correct beliefs through the search tree (assuming the correct prior distribution is used). The approach to inference introduced in this chapter takes advantage of this fact by calculating beliefs from statistics in the ISMCTS tree, which are used as a prior in subsequent searches. In Section 7.4 bluffing behaviour is induced in the ISMCTS algorithm through the use of self-determinizations, which are determinizations of the search players own hidden information (from the opponents point of view). This prevents ISMCTS from learning to exploit the root player's own hidden information which was shown to be an issue in Chapter 6, and allows ISMCTS to learn the utility of bluffing moves. The approach to performing inference and bluffing with ISMCTS developed in this chapter, improves the strategic performance of ISMCTS without requiring domain knowledge. This approach does not converge to a Nash-equilibrium, yet still offers a significant improvement over ISMCTS when bluffing and inference are important aspects of a game.

The structure of this chapter is as follows. Firstly Section 7.1 introduces the concepts of bluffing and inference in games and outlines how this can be achieved using MCTS as an opponent model. Section 7.2 details the modifications made to the MO-ISMCTS algorithm used for experiments in this chapter. In Section 7.3 it is shown how MCTS can be used as an opponent model for inference,

and Section 7.4 introduces several techniques that can induce bluffing behaviour in the ISMCTS algorithm. Section 7.5 presents experiments which evaluate the effectiveness of the modifications made to ISMCTS in this chapter in three games, The Resistance, Scotland Yard and Saboteur. Finally Section 7.6 summarizes the results of experiments and discusses how ISMCTS could be further improved in these games.

7.1 Background

In this section it is discussed how bluffing and inference may be learned using ISMCTS. Firstly the following terms are defined in the context of this work:

- *Determinizaton:* A selection of a state from an information set, which is treated as a game of perfect information.
- *Opponent Model:* A mapping from the information sets belonging to an opponent to a probability distribution over actions available in each information set.
- *Inference Model:* A mapping from an opponent model and a set of observed opponent actions to a probability distribution over states in an information set.
- *Feature:* A mapping from states to elements of a finite set. For example a feature could be the identity of the player holding the Ace of Spades, or the number of Spades held by a particular player.
- *Probabilistic Reasoning:* Given a distribution over states within an information set and a set of features, the values of features can be treated as random variables. It can be useful to design rules which influence play depending on the expected value of features (a process often described as inference). Probabilistic reasoning does not need to be coupled with an Inference Model. In the absence of an Inference Model, the expected value of features can be estimated by assuming all states are equally likely.

A determinization of an information set is a selection of a state within an information set, assumed to be the true state of a game and to be subject to analysis. Determinization methods have been successful in a wide variety of games with hidden information, but have several inherent flaws. Notably Frank and Basin [28] identify the issues of strategy fusion and non-locality. Previous work described in Chapter 5 has identified that the effect of strategy fusion is more significant in Dou Di Zhu than the lack of a good opponent model.

However the ISMCTS algorithm still suffers from the problem of non-locality. Determinizations are sampled uniformly from the set of all legal determinizations by ISMCTS, however this assumption of uniformity is incorrect. In the worst case it may be deduced that some legal determinizations are actually impossible if it is assumed the opponents behave rationally in the game-theoretic

sense, for example determinizations that are only possible if an opponent chose not to play a move leading to an immediate win earlier in the game. In this case it may be concluded that the true state of the game is restricted to an information set where the winning move was unavailable to the opponent on a previous decision. More generally if it is assumed that the opponents play rationally, then some states within an information set may be more likely than others.

This more general case can be described as applying *inference*, specifically inference applied to an opponent model. An opponent model is any technique which can be used to estimate the likelihood of an opponent taking a particular action in a particular state. After observing an opponent take an action, an opponent model can be used to determine which states are more or less likely, producing a probability distribution over states within an information set. When game players discuss strategy, the term "inference" sometimes takes on a wider meaning and includes probabilistic reasoning independent of an opponent model, for example the probability an opponent holds an Ace given that the other three Aces have already been seen. However in this work inference is considered specifically to be the process of exploiting an opponent model to deduce the likelihood of certain states, rather than exploiting the likelihood of certain states to improve decision making.

There is value to applying inference in a game when the following two criteria can be satisfied:

- It can be deduced by some means that states within an information set are more or less likely than each other, and this deduction is informed by an assumption of rational play or by some other opponent model
- Knowing which states are more or less likely has some influence over which decisions are made

When these conditions are satisfied often enough, applying inference enables a player to exploit their opponent or force them to hide information more effectively. For games where bluffing and inference are important tactics this should lead to stronger play. Furthermore work applying inference to the game Spades helped improve the plausibility of the AI decision making [10]. The methods discussed in this work focus on satisfying the first criteria, since ISMCTS is capable of altering decisions based on the distribution of determinations used. That is ISMCTS is capable of probabilistic reasoning, since the random determinization process ensures that the correct belief distribution is propagation to every node in the tree [82]. This allows ISMCTS to exploit the distribution of states at each node in order to improve decision making, therefore ISMCTS can already alter decisions based upon an inference model if it is used to alter the probability distribution determinizations are sampled from.

7.1.1 Existing Approaches

The most common theoretical framework for inference in games is *Bayesian inference*. Consider a state s in the current information set. There is a *prior*

belief $P(s)$, the probability that s is the actual state of the game. Upon observing an opponent action a, the *posterior belief* $P(s|a)$ is the probability that s was the actual state of the game given that a was played from that state. If it is assumed that the opponent's (mixed) policy is known, then $P(a|s)$ is the probability that the opponent plays action a from state s. Given the prior belief and the opponent policy, Bayes' theorem can be applied to obtain the posterior belief:

$$P(s|a) = \frac{P(a|s)P(s)}{P(a)} \tag{7.1}$$

where

$$P(a) = \sum_{u \in [s]^{\sim_1}} P(a|u)P(u) \tag{7.2}$$

is a normalising constant.

There exist several approaches to integrating inference (Bayesian or otherwise) with MCTS for particular games. In practice the opponent's policy is not known, so it is approximated by an *opponent model*. Most existing approaches for applying inference with MCTS use an opponent model which is computed offline, but used to determine a distribution over states at runtime. One advantage to this approach is that if the opponent model is accurate MCTS is capable of learning to exploit it. For example Ponsen et al [89] learn an opponent model for Poker which is used to influence the cards dealt and select actions for the opponent. Buro et al [49] apply a similar method to the card game Skat, where a learned opponent model is used to estimate the likely cards held by each player. Previous work on Spades [10] uses a knowledge base of hard-coded rules (parametrised by weights tuned offline) to influence the distribution of cards dealt to players depending on factors such as their bid.

There has been less work on methods which compute an opponent model online, though there is a natural synergy between this approach and MCTS since MCTS already models the decisions of opponents. If it is assumed that MCTS is a good opponent model, an inference model can be built by using tree statistics. For example if every state in an information set is sampled sufficient times, it is possible to measure the proportion of times each action was selected for an opponent from each state and update the belief state using Bayes' theorem. However in practice the number of states is often too large for this approach to be tractable and instead an approximate model must be used (which is beyond the scope of this work). This approach has been successful in applications such as Scrabble [90]. Silver and Veness [82] propose an approximate method using a particle filter, which approaches a correct belief state as more particles are used. One drawback of this method is that particle deprivation may occur as the particles disperse in a large state space and a particle reinvigoration method is needed. Particle filtering is discussed further in Section 7.3

7.2 ISMCTS with multiple opponent trees

The version of ISMCTS used in this chapter is based on the MO-ISMCTS algorithm described in Chapter 6. MO-ISMCTS models the differing viewpoints of the players by constructing several trees (one per player) in parallel. This idea can be extended by constructing several trees per player, with a tree for each opponent information set). Each iteration samples a determinization from the root information set, and selects the corresponding tree for each player. For example in a hidden role game such as The Resistance or Saboteur, each player has one tree for each possible role they may have, thus allowing ISMCTS to model the different policies followed by the opponent depending on their role. A determinization specifies the roles of the players, and the corresponding trees are descended. Pseudocode for this variant of ISMCTS is given in Algorithm 6.

This tree representation ensures that there will be a unique node for every unique information set. There will be also many other non-decision nodes in the trees which exist to correctly transition between decision making nodes, given each players observations of moves during a playout. Creating lots of extra nodes which are not used to store useful statistics appears highly memory inefficient and a more compact representation could avoid building trees by only storing the decision nodes (corresponding to information sets). However in this case extra logic must be introduced to calculate which information set node to transition to after an action is chosen, which would require storing enough information to distinguish between each information set the algorithm has stored statistics about. The software framework used in this work only stores statistics in nodes and no information about the states the nodes represent. There structure in the trees which are generated by multi-tree ISMCTS and the non-decision nodes allow the transitions between information sets to be calculated efficiently at run time, by simply following the appropriate edges in the trees. This is exploiting the fact that in most games, an action typically contains a small amount of information compared to a full game state.

The drawback of utilising additional trees is that the ISMCTS learns slower, since not every tree is updated on each iteration, but the additional trees improve the opponent model of ISMCTS which is particularly important if the opponent model is to be exploited for inference. Therefore the additional trees can be seen as a trade-off between a fast learning rate and a better opponent model. It is likely that the learning rate can be increased by utilising enhancements that share knowledge between trees such as the information capture and reuse enhancements introduced in Chapter 8.

For experiments in this Chapter the implementation of ISMCTS uses UCB-Tuned [39] for selection, which has been shown to outperform UCB1 in several domains [44, 40] without requiring an exploration constant to be tuned, and thus is arguably a better default choice of selection policy. Preliminary experiments showed a marginal benefit (around a 3–4% increase in win rate) from using UCB-Tuned instead of UCB1 in the game The Resistance.

Algorithm 6 The MO-ISMCTS algorithm extended to have multiple trees per opponent

function ISMCTS($[s_0]^{\sim_1}, n$)
  for each player $i = 0, \ldots, \kappa$ and each possible determinization d create a single-node tree with root u_d^i, if a tree with this root does not already exist
  for n iterations **do**
    construct a random determinization d and use only nodes/actions compatible with d this iteration
    for this iteration, "the player i tree" is the tree rooted at u_d^i, and "all trees" means the player i trees for $i = 0, \ldots, \kappa$

    // Selection
    repeat
      descend all trees in parallel using a bandit algorithm on player ρ's tree whenever player ρ is about to move
    until nodes $v^0, \ldots, v^\rho, \ldots, v^\kappa$ are reached in trees $0, \ldots, \kappa$ respectively, player ρ is about to move at node v^ρ, and some action from v^ρ leads to a player ρ information set which is not currently in the player ρ tree **or until** v^ρ is terminal

    // Expansion
    if v^ρ is nonterminal **then**
      choose at random an action a from node v^ρ that is compatible with d and does not exist in the player ρ tree
      for each player $i = 0, \ldots, \kappa$ **do**
        if there is no node in player i's tree corresponding to action a at node v^i, then add such a node

    // Simulation
    run a simulation from v^ρ to the end of the game using determinization d, (starting with action a if v^ρ is nonterminal)

    // Backpropagation
    for each node u^i visited during this iteration, for all players i **do**
      update u^i's visit count and total simulation reward
      for each sibling w^i of u^i that was available for selection when u^i was selected, including u^i itself **do**
        update w^i's availability count

  return an action from $A([s_0]^{\sim_1})$ such that the number of visits to the corresponding child of the node for $[s_0]^{\sim_1}$ is maximal

7.3 Inference using MCTS statistics

The idea of *particle filtering* in an ISMCTS-like algorithm was suggested by Silver and Veness [82]. The idea is to record the frequency with which each determinization (particle) reaches each node in the tree, and use this to update the belief distribution in response to observed moves.

Consider a tree with root node u and a child v corresponding to action a. Consider also a determinization d for the root information set where d is chosen with probability $P(d)$, the prior belief that the current state is d. Then the number of visits to child v with determinization d as a fraction of the total number of visits to v is an estimate of $P(d|a)$: the probability that d was the current determinization given that action a was selected from the root. Hence this method allows the posterior belief to be sampled empirically without need for Bayes' rule (Equation 7.1).

In many games, there are too many states per information set to enumerate. Silver and Veness [82] sample a small number of determinizations to use as particles, and introduce a *reinvigoration* mechanism to counteract the depletion of the particle set as the tree is descended. Another approach is to perform inference on features of a state (similar to Buro et al [49]), but doing so would require an assumption of independence between features. Depending on how reasonable an assumption this is, the method may still give a good enough approximation of the belief distribution over states. There are a small enough number of information sets in The Resistance that no reduction was necessary for the experiments in this chapter (likewise for the experiment on Scotland Yard). In the game Saboteur, a reduction was made by ignoring the cards held in each players hand (any set of cards in hand was considered to be in the same information set), which is reasonable since inferring the identity of a player (which is fixed) is more strategically important than learning the contents of their hand (which changes rapidly).

Example 1. In The Resistance or Saboteur, determinizations are sampled from the possible sets of spies or saboteurs; in Scotland Yard deteriminzations are sampled from the possible locations of Mr X. In The Resistance and Scotland Yard a determinization corresponds to a unique tree for each player. In Saboteur this is not the case, as the state carries a lot more hidden information than the player identities (the cards in players' hands, the shuffled order of the deck, and the location of the gold), so after determinizing the set of saboteurs (and selecting the corresponding search trees) the other hidden information is determinized.

To perform inference within ISMCTS *particle filter inference* is used where the particles are determinizations. Recall Algorithm 6: the player's decision tree is rooted at node $[s_0]^{\sim_1}$ (assuming without loss of generality that player 1 is the root player). For each node u^1 in this tree, a count $c(u^1, d)$ is maintained of the iterations passing through that node for which the current determinization was d. During backpropagation, at line 23, if $i = 1$ then $c(u^i, d)$ is incremented by 1 (where d is the determinization chosen on line 4).

The player also maintains a belief distribution ϕ_1 over determinizations, separate from the tree. At the beginning of the game, this distribution is chosen to match the distribution of initial game states (which often means a uniform distribution). The belief distribution is updated as in Algorithm 7. The update has two steps: a game-specific *propagation* step, and a game-independent *inference* step. In Bayesian terms, the propagation step maps the posterior distribution at time t to the prior at time $t+1$, which the inference step then maps to the posterior at time $t+1$.

The propagation step is based on the following question: if a state has some determinized hidden information, and move m is applied, what is the value of the hidden information in the resulting state? In some games the information never changes once fixed: for example in The Resistance, no move ever changes the identities of the spies. In this case the propagation step leaves ϕ_1 unmodified. In Scotland Yard however, if move m is a move by Mr X, then the hidden information (the location of Mr X) will change. Here ϕ_1 is updated by considering all actions in the partially observable move m to be equally likely.

Example 2. Suppose that in Scotland Yard $\phi_1(11) = \frac{1}{4}$, $\phi_1(12) = \frac{3}{4}$, and m is a Mr X move by taxi. (Recall that determinizations for Scotland Yard are locations for Mr X, so $\phi_1(11)$ is the probability that Mr X is in location 11.) From node 11, Mr X could have moved to node 3, 10 or 22. From node 12, Mr X could have moved to node 3 or 23. Thus set:

$$\begin{aligned}
\phi_1'(3) &= \frac{1}{3}\phi_1(11) + \frac{1}{2}\phi_1(12) = 0.458\dot{3} \\
\phi_1'(10) &= \frac{1}{3}\phi_1(11) = 0.08\dot{3} \\
\phi_1'(22) &= \frac{1}{3}\phi_1(11) = 0.08\dot{3} \\
\phi_1'(23) &= \frac{1}{2}\phi_1(12) = 0.375
\end{aligned}$$

The propagation step also takes care of hard constraints: if move m is not legal in determinization d, then $\phi(d)$ is set to zero. For example in The Resistance, if move m is the environment move "reveal that there was one sabotage card on the mission", and x is a set of spies that does not include any members of the mission team, then set $\phi(x) = 0$. If any probabilities are set to zero, the rest of the probabilities in ϕ are normalised.

The rest of Algorithm 7 describes the inference step. Initially u^1 is the root of player 1's most recent decision tree, i.e. $[s_0]^{\sim_1}$. The belief distribution is shifted towards the frequency distribution of determinizations that visited the node corresponding to m. This shift is weighted by w, the number of visits to the node in question as a fraction of the total search iteration budget. The idea here is that infrequently visited branches should not bias the distribution too much, as otherwise the inference may become brittle. Also since $w < 1$ (unless only a single move was available), the frequency distribution never completely

Algorithm 7 Belief distribution update for particle filter inference.

```
1: function ObserveMove(m)
2:     update φ1 uniformly according to m (propagation step) setting φ1(d) = 0
   if move m is not legal in determinization d
3:     normalize φ1
4:     let v^1 be the child of u^1 corresponding to move m
5:     if m was not played by player 1 or by the environment player, and was
   not the only move available then
6:         let w be the v^1 visit count divided by the total number of iterations
7:         for each possible determinization d do
8:             update φ1(x) ← (1 − w)φ1(x) + w c(v^1,d) / v^1 visit count
9:
10:    // Descend the tree
11:    let u^1 ← v^1
```

replaces the belief distribution, so a determinization probability that starts as nonzero will stay nonzero (unless explicitly set to zero during the propagation step).

The belief distribution ϕ_1 is used to sample determinizations for subsequent decisions. For The Resistance and Scotland Yard the sampling function is straightforward: it simply clones the actual root state and overwrites the spy identities or Mr X's location with the appropriate value. In Saboteur the sampling function chooses the identities of players according to ϕ_1 as well as additionally randomizing randomizing the location of hidden cards and the gold.

7.4 Self-determinization and bluffing

Bluffing (and information hiding) requires *self-determinization*: consideration of states that are outside the player's current information set but in an opponent's information set. This models the uncertainty an opponent has about the player's information and allows the player to exploit that uncertainty. ISMCTS can use self-determinizations without affecting the player's ability to plan ahead: iterations using determinizations outside the player's information set do not descend or update the decision tree, but the fact that they descend and update the opponents' trees as usual means that they do factor into the opponent model. However a significant amount of time is potentially "wasted" considering lines of play that are known to be impossible, for the sake of more accurate opponent modelling. Thus a balance must be struck between searching "true" determinizations (in the root information set) and self-determinizations (not in the root information set). This gives rise to a new problem: any bias towards true determinizations will also influence the opponent trees, causing the hidden information to "leak" into the modelled opponent policy and thus lessening the effectiveness of self-determinization.

A self-determinizing player needs two belief distributions: the usual belief

distribution ϕ_1 used to select true determinizations, and a distribution ψ_1 used to select self-determinizations. The latter is the player's model of what the opponent's have inferred so far. Both are updated as in Algorithm 7, however ψ_1 is initialised and updated without knowledge of the player's hidden information. In The Resistance for example, ϕ_1 is initialised by setting the probabilities of spy configurations containing the player to 0 (if the player is not a spy) or the probability of the actual spy configuration to 1 (if the player is a spy), whereas ψ_1 is initialised by setting all spy configurations to equal probability, regardless of whether they contain the player and whether the player is a spy. In Saboteur ϕ_1 and ψ_1 are initialized in the same way with regards to saboteur configurations except if the player is a saboteur, all saboteur configurations where the player is a saboteur have equal probability in ϕ_1 (and 0 probability otherwise). In Scotland Yard only Mr X needs to self-determinize, so ϕ_1 is initialised by setting the probability of the actual location of Mr X to 1 and ψ_1 by giving each starting position for Mr X equal probability (although an expert player may determine that some starting positions are less likely than others).

The distribution ϕ_1 is updated according to the player's decision tree. If ψ_1 is updated using the decision tree, information about the player's hidden information would leak into ψ_1 and the benefit of self-determinization would be defeated. Instead ψ_1 is updated by merging all the trees from the player's perspective, including the decision tree and all trees for self-determinizations. The visit counts in the merged tree are obtained by summing the visit counts across all trees. This can be thought of as a tree "averaged" across all self-determinizations, or the hypothetical decision tree that would be built by a player whose beliefs are captured by ψ_1 (an external observer who cannot see any of the players' hidden information).

In many games (including Resistance and Scotland Yard), bluffing is a strategic decision. That a spy should conceal his identity or Mr X should conceal his location seems obvious to a human player, but to discover this purely by tree search is difficult, as the consequences of revealing the information are usually not instantaneous. This suggests that game-specific knowledge is required to encourage bluffing. Indeed, injecting such knowledge into MCTS was shown by Nijssen and Winands [72] to be successful for Scotland Yard: there the moves of Mr X are biased towards those which increase the number of his possible locations from the seekers' point of view.

One of the strengths of online approaches such as MCTS is that they exploit the Markovian property of perfect information games to ignore parts of the game tree that are not in the subtree rooted at the current state. The player can forget not only the line of play that led to the current state, but also the alternate lines of play that were available but not taken. In imperfect information games this Markovian property breaks down, and bluffing explicitly requires consideration of other parts of the game tree to avoid non-locality. Hence offline techniques such as MCCFR [120, 121] or Multiple MCTS [101], which consider the whole tree from the initial state of the game, may be better suited as a general-purpose technique for games where bluffing is important — the major disadvantage being that these techniques are not suitable for games with combinatorially large trees.

This is discussed in more detail in Chapter 3.3.4 page 32. In this work it is shown that bluffing behaviour can emerge from ISMCTS for the game The Resistance, at the cost of a slower learning rate for ISMCTS. However it should be possible to a achieve this result in larger combinatorial games using a combination of increased computational resources and domain knowledge injection.

A number of approaches to self-determinization in ISMCTS are tested:

Pure. Simply run ISMCTS with determinizations sampled from ψ_1, i.e. all self-determinizations. Some of these determinizations will happen to be true determinizations and thus update the decision tree, but otherwise no special effort is made to search the decision tree.

Split. Perform two ISMCTS searches, one after the other, retaining all trees from the first to the second. The first samples determinizations from ψ_1, the second from ϕ_1. The first search populates the trees with statistics from self-determinizations, whereas the second searches only true determinizations but carries the tree statistics over from the first step. The second search has the potential problem that hidden information leaks into the opponent trees during the second step.

Two-step. As the split method, but the second search does not update the opponent trees, to counteract the leakage of information. To make the tree policy nondeterministic, selection in opponent trees uses a mixed policy where the probability of selecting a child node is proportional to the number of visits it received in the first step.

Bluffing. As the split method, but with a different mechanism for choosing the move to play at the end of the search (rather than the convention of choosing the most visited move). Denote the mean reward for a move m from the root of the decision tree by μ_m and the standard deviation by σ_m. Choose m^* with maximal number of visits from the root of the decision tree, as usual. Let

$$M^* = \{m \ : \ \mu_{m^*} - \mu_m < \min(\sigma_{m^*}, \sigma_m)\} \ , \tag{7.3}$$

i.e. M^* is the set of moves whose average reward is within one standard deviation of the reward for the best move. Now for each move in M^*, sum the number of visits from the root across all the current player's trees (i.e. the decision tree and all trees corresponding to self-determinizations), and play the move for which this sum is maximal. In other words, the chosen move is the most visited move across all self-determinizations that is not significantly worse than the most visited move in true determinizations; the bluff that is not significantly worse than the best non-bluff.

The pure method is the most theoretically sound method and similar to Smooth UCT, which can converge to a Nash-equilibrium [84]) since the player models will not be able to learn to exploit inferences they have not made, however this method "wastes" most of its iterations by not updating the actual

decision tree. The split method provides a compromise by spending at least half its time searching true determinizations, but risks leaking information into the opponent model. The two-step method prevents this leakage of information by not updating the opponent model at the end of the first step, at the cost of the opponent model being weakened due to having fewer iterations. Any bluffing behaviour observed in these three methods is an emergent property of the search. The bluffing approach explicitly integrates bluffing behaviour into the move selection, by choosing a move which is plausible when the opponent model "cheats" (searching true determinizations) but was preferred over other moves when the opponent model does not "cheat". In other words, amongst the moves which are the most robust against opponent's inference, choose the move which best exploits their lack of knowledge.

7.5 Experiments

This section presents experiments which evaluate the effectiveness of the modifications to ISMCTS described earlier in this chapter across three games: The Resistance, Saboteur and Scotland Yard.

7.5.1 Inference and self-determinization methods for The Resistance

In this section the performance of various ISMCTS players for The Resistance is tested. Two non-spy players are tested: one sampling determinizations uniformly at random, and one using the particle filter inference mechanism described in Section 7.3. Both of these players use 20 000 iterations per decision, and neither player uses self-determinizations. For the spies, a "True" player without self-determinization (using 20 000 iterations per decision) is tested, as well as players using each of the variants of self-determinization in Section 7.4. The self-determinizing players use 40 000 iterations per decision; the three players based on the "split" method perform 20 000 iterations with self-determinizations and 20 000 with true determinizations. For each tested configuration 1000 games were played, each with 5 players: each of the 10 possible spy configurations was tested 100 times, each time with all the spy players using one algorithm and all the non-spy players using another.

Results shown in Figure 7.1 indicate that inference is extremely powerful against a non-self-determinizing player, reducing the latter player's win rate by 66.4%. The pure, split and two-step methods are around twice as strong as the non-self-determinizing player, with no significant difference between the three. The most effective counter to the inference player is the bluffing player, with an improvement of 31.7% over the non-self-determinizing player. This shows that this bluffing method counteracts roughly half of the advantage gained by the non-spy players performing inference, and results in a more evenly matched game.

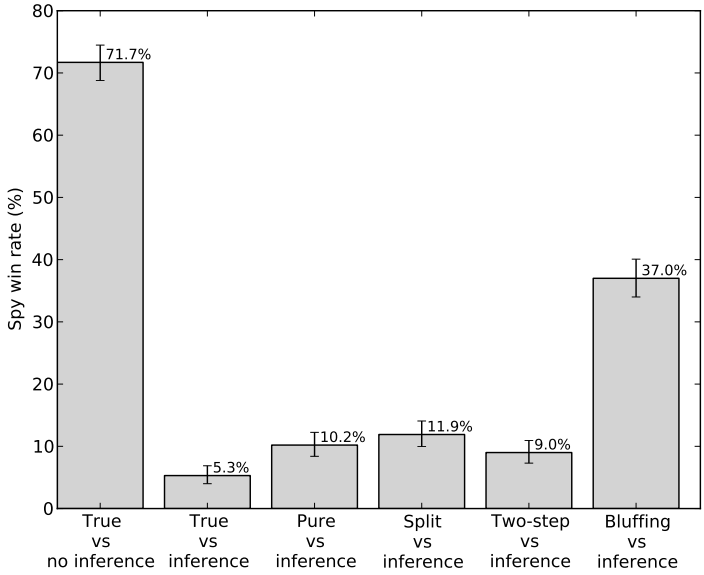

Figure 7.1: Comparison of inference and self-determinization methods for The Resistance. Each bar shows the win rate for algorithm playing as the Spy team against non-spies using ISMCTS with or without particle filter inference.

7.5.2 Balancing true and self-determinizations

In the previous section, the split, two-step and bluffing players all used an equal division of their iteration budgets between self-determinizations and true determinizations. This section investigates the effect of this split on the player's performance. Once again the particle filter inference non-spy player using 20 000 iterations played against the split spy player using 40 000, but here the number of iterations T_1 used in the first (self-determinizing) search is varied between 0 and 40 000, with $T_2 = 40\,000 - T_1$ iterations for the second (true determinization) phase. Note that $T_1 = 40\,000$ is equivalent to pure self-determinization, and $T_1 = 0$ is equivalent to a player which uses only true determinizations.

Results are shown in Figure 7.2 and indicate an upward trend as the number of self-determinizing iterations increases, with performance reaching a plateau between 15 000 and 35 000 self-determinizing iterations and dropping off for 40 000 iterations. There is a trade-off to be made between searching the decision tree and searching other self-determinizations; these results suggest that devoting anywhere between $\frac{3}{8}$ and $\frac{7}{8}$ of the total search budget to self-determinizations yields reasonable performance. The experiment suggests that 5000 true determinizaions is a sufficient amount and any additional determinizations sampled should be self determinizations.

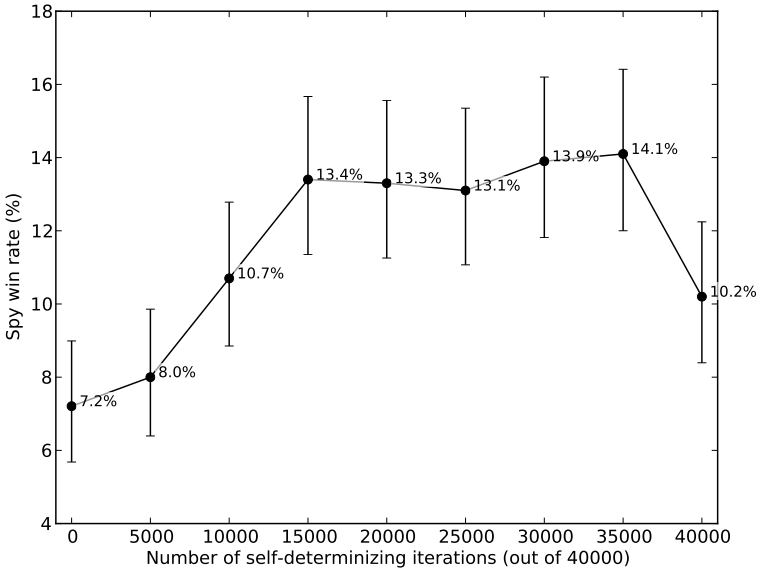

Figure 7.2: Performance of the "split" spy player for The Resistance, devoting various numbers of iterations to self-determinizations.

7.5.3 Emergence of bluffing

Figure 7.3 shows the performance of the true, pure, split and bluffing players with varying numbers of ISMCTS iterations, for spy players in The Resistance. In each case the non-spy players use inference but no self-determinization. All players (spies and non-spies) use the specified number of iterations, with the split and bluffing spy players using half the iterations for self-determinizations and half for true determinizations. The four spy player types use the same number of iterations in total, unlike the experiments in Section 7.5.1 where the self-determinizing players have twice the budget of the pure player. Note that the number of iterations varies for all players, so the spies' win rate can fluctuate up or down depending on which team is improving more quickly as the number of iterations increases.

For small numbers of iterations the performance of the different spy players is close, but the true and split players are stronger than the bluffing player by a statistically significant margin (at 95% confidence). Between 5 000 and 200 000 iterations, the bluffing player is much stronger than the other two. However for very large numbers of iterations the split player overtakes the bluffing player whilst performance of the true player remains poor. This suggests that the split player with sufficiently many iterations can produce bluffing behaviour without an explicit bluffing mechanism. However it should be noted that the computational resources required for this player might be infeasible for a commercial application: an efficient C++ implementation of a player using 1 000 000 iter-

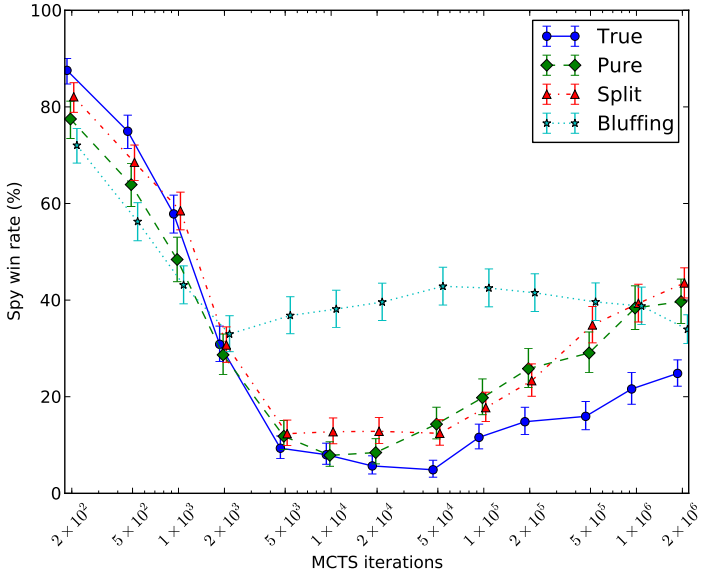

Figure 7.3: Performance of ISMCTS variants for The Resistance for varying numbers of iterations. Non-spy players use ISMCTS with particle filter inference, whilst spies use ISMCTS with the specified self-determinization method (win rates are given for the Spy team). Both spies and non-spies use the specified number of iterations per decision.

ations takes around 10 seconds per decision on a desktop PC with a 2.53GHz Intel Xeon processor and consumes around 1GB of memory.

An upward trend is visible in the win rate for the true player for more than 50 000 iterations, albeit a slower one than for the split player. One possible explanation for this is that the non-determinizing spy player may pessimistically assumes that the game is lost, as it is assuming that the non-spies know that it is a spy. MCTS tends to play randomly when its decisions do not affect the result of the game. However, playing randomly reduces the ability of the non-spies to perform inference, hence possibly leading to a situation where the spies can win. It Chapter 6 it was observed that large numbers of ISMCTS iterations can lead to overly pessimistic assumptions in the Phantom $(4, 4, 4)$ game.

7.5.4 Effect of inference on the belief distribution

Figure 7.4 shows, for The Resistance, how the non-spy player's belief distribution evolves over time when using particle filter inference. In particular, the plot shows the probability assigned to the actual configuration of spies in the game. For comparison, Figure 7.6 shows how the belief distribution evolves if the player performs no inference and only keeps track of hard constraints. The inference player displays an overall upward trend much steeper than that for the player without inference and very rarely is the probability lower than the initial value (which would imply that an incorrect inference was made).

Figure 7.4 suggests some correlation between the success of inference and the outcome: in many of the games won by the spies, the non-spy players assign a relatively low probability to the actual spy configuration. In this experiment the spies are not using any self-determinizing techniques, so are not deliberately trying to mislead the inference engine in this way. Figure 7.5 shows the results when the spy players use the bluffing technique described in Section 7.4. Bluffing results in the probability of the actual spy configuration being typically lower and results in the spies winning more games. In particular, when the spies bluff there was no instances where the actual configuration had probability 1 after the first mission (which would happen if both spies were on the mission and chose sabotage for example). There are also many games won by the spies on the last mission whereas there are none when the spies do not bluff.

Figure 7.4 also demonstrates the overall strategy used by the ISMCTS players: the team choice for the first mission is almost always voted down four times to allow the fifth player the final choice. This is visible in the plot as a sequence of four distinct "steps" between $x = 0$ and $x = 1$. From these upward steps, it can be seen that the non-spy players are able to infer useful information from the team choices and votes in this first round. It is worth observing that, while this strategy differs from that used by some groups of human players, the "best" strategy for the first round is a frequently debated topic [1].

In Figure 7.6 some trajectories reach a probability of 1 for the actual spy configuration, meaning that the player knows the identity of the spies. If both

[1] See e.g. `http://boardgamegeek.com/thread/640992/is-the-first-mission-pointless-for-5-6-or-7-playe`

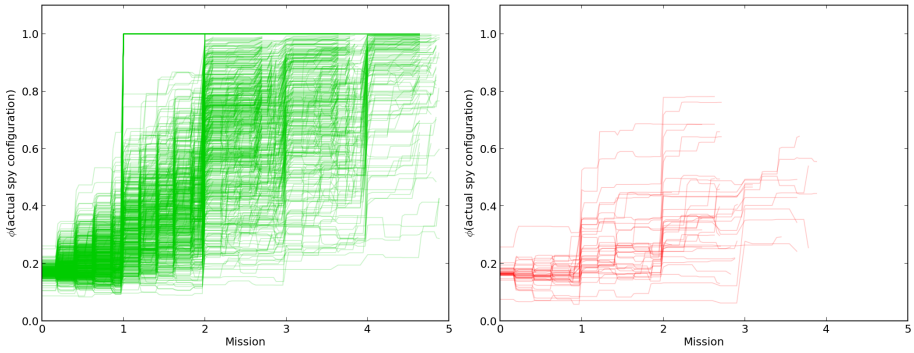

Figure 7.4: Plots of the inferred probability for the correct spy configuration across 250 games of The Resistance. Time moves left to right along the x-axis, from the beginning to the end of the game, scaled to align mission numbers. The left-hand plot shows games where the non-spy team won; the right-hand plot shows games won by the spies.

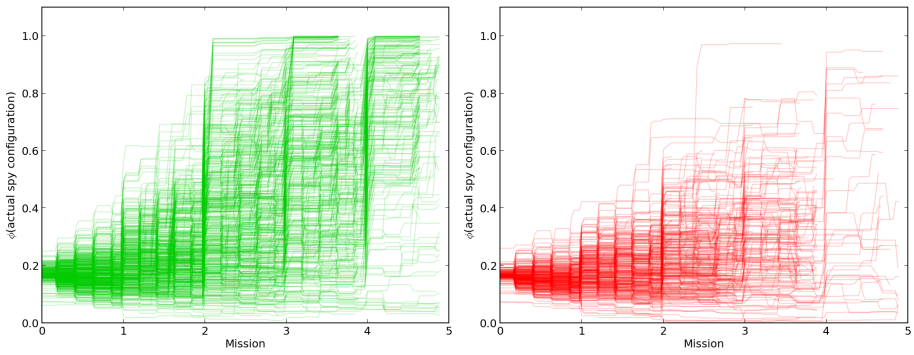

Figure 7.5: As Figure 7.4, but with bluffing spy players.

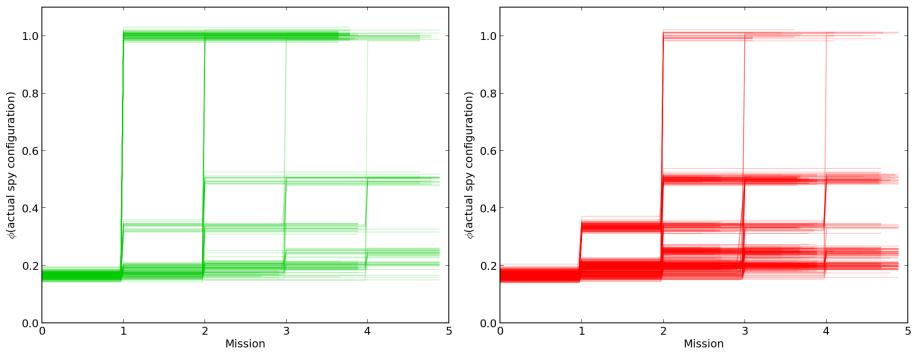

Figure 7.6: As Figure 7.4, but without particle filter inference, i.e. only keeping track of hard constraints. To show the density of overlapping lines, a small random vertical offset has been added to each line.

spies are on a 2-person mission and both sabotage, this reveals their identities to the other players. On a 3-person mission this potentially only reveals the spies' identities to the third player on the mission, with the other players not knowing which of the three is not a spy. This information asymmetry between non-spies explains why the spies do not always lose after revealing themselves to another player in this way (if the spies are revealed to all players, the non-spies can always force a win by refusing to choose spies for missions, and using their majority to vote down any missions chosen by spies).

7.5.5 Application to other games

The Resistance is an ideal test case for the methods described in this chapter: bluffing and inference are the main mechanics of the game, the number of states per information set is small, and the game tree is too large to solve analytically but small enough to be tractable to MCTS without knowledge-based enhancements. The scalability of these methods to two more complex games: Scotland Yard (see Chapter 4.2.12 page 48) and Saboteur (see Chapter 4.2.13 page 49) is investigated in this section.

Figure 7.7 compares the performance of ISMCTS players for Scotland Yard. The experimental conditions are as in Section 7.5.1, with the exception that the "spy" player algorithms are now applied to Mr X, the "non-spy" algorithms to the player controlling the seekers, and the initial conditions (the starting locations of Mr X and the seekers) are chosen randomly for each game. The benefit of inference for the seekers is more modest than for The Resistance, but still statistically significant. The pure and split self-determinization methods are similarly beneficial for Mr X, but two-step self-determinization and bluffing are not.

The game tree for Scotland Yard is much larger than that for The Resistance

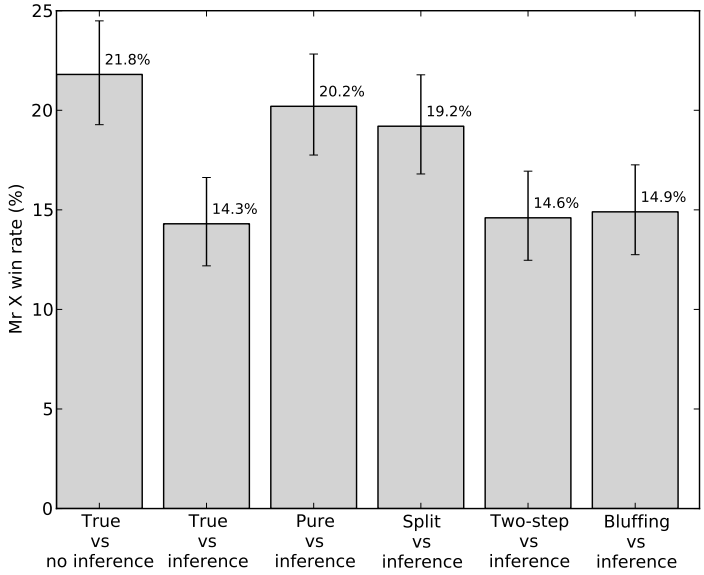

Figure 7.7: Comparison of inference and self-determinization methods for Scotland Yard showing the win rate of each algorithm playing as Mr. X against seekers using ISMCTS with or without particle filter inference

due to a higher branching factor. This makes it more difficult for MCTS to find good moves without large numbers of iterations or heuristic knowledge. Two-step bluffing devotes only half of its iterations to learning the opponent policy, which is likely to impair the quality of the player's own policy. Bluffing potentially causes the player to play suboptimal moves, even if the search is able to find the best move. Scotland Yard also has many more states per information set than The Resistance (one for each possible Mr X location), and the strategy in terms of information gathering and hiding has more to do with the degree of uncertainty (i.e. the number of possible Mr X locations) than whether Mr X is more likely to be in one location than in another. The work on Scotland Yard by Nijssen and Winands [72] exploits this in designing heuristic knowledge for Scotland Yard. In The Resistance, the degree of uncertainty is small and more-or-less fixed: each player either is or is not a spy.

As in The Resistance, in Saboteur each player has a hidden role. In The Resistance the hidden roles are the only source of hidden information, but in Saboteur each player also has cards in hand, knowledge of which cards they have discarded, and possibly knowledge of one or more goal cards. ISMCTS players which "cheat" (have access to all hidden information except the other players' roles) were tested against ISMCTS players which do not, and found no significant advantage to the former over the latter. This suggests that learning the identities of other players and hiding your own is more strategically important than trying to deduce the cards held by other players. Hence the inference

model can safely ignore this extra hidden information and consider only the player roles. In any case, it would be very difficult to arrive at meaningful inferences for the cards in hand: each turn results in a card being played or discarded and another being drawn from the deck, so the information is likely to change faster than it can be inferred.

For Saboteur, ISMCTS constructs two trees per player: one for the player as a gold-digger and one as a saboteur. Note that unlike for The Resistance and Scotland Yard, this means that each tree covers several information sets for that player with each tree collecting statistics for many possible hands of cards. Saboteur is a good test case for the scalability of this approach, since the degree of uncertainty in the player roles is small and fixed, inference and bluffing with respect to these roles are key mechanics, and the game tree is very large and stochastic.

For Saboteur, an ISMCTS player using inference was tested against four players not using inference. The inference player's seat was randomised as were the identities of the saboteurs, so an evenly matched player would have an expected win rate of 50%. Over 1000 trials, the inference player achieved a win rate of 54.9% ($\pm$3.2% at 95% confidence). The benefit of inference is smaller than for the Resistance but statistically significant. Examining plots of the inferred probability for the actual saboteur identities (Figure 7.8; compare with Figures 7.4), it can be seen that inference does often produce a bias towards the correct configuration but that the effect is modest. The large branching factor means that the iterations are spread more thinly amongst the available moves, so the factor w in Algorithm 7 is typically small. A more aggressive belief distribution update (for example with an artificially boosted value of w) may produce a stronger effect, but also runs the risk of reinforcing erroneous inferences when the number of iterations is insufficient.

Another reason for the relatively small benefit of inference in Saboteur is that ISMCTS without knowledge injection and a sufficient number of iterations is not very strong and so does not produce good opponent models for decision nodes deeper in the tree. It is likely that heuristic knowledge would be required to obtain plausible play: the branching factor is large, but many of the available moves are quickly rejected by human players as being inconsequential or obviously bad. Given the small benefit of inference, self-determinization methods were not tested for Saboteur: the goal of these methods is to reduce the advantage to the opponent's of inference, and the already small size of this advantage would make it difficult to obtain statistically significant results. It is expected that using a large number of iterations or injecting knowledge into ISMCTS would improve the quality of the opponent models and produce results similar to those for The Resistance and Scotland Yard. The information capture and reuse methods introduced in Chapter 8 would likely improve the learning rate of ISMCTS in these games.

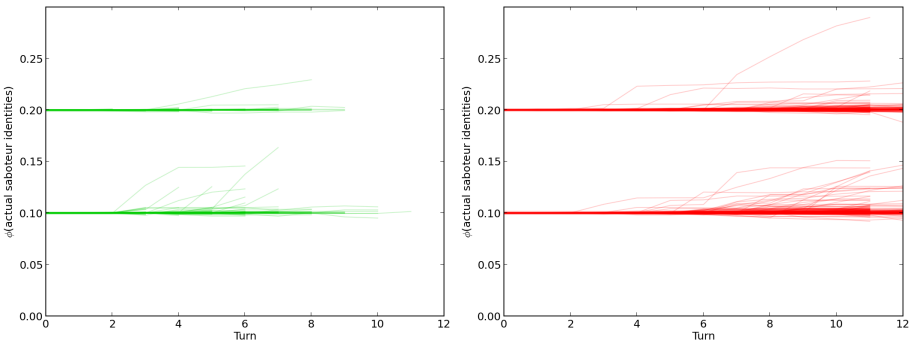

Figure 7.8: Plots of the inferred probability for the correct saboteur configuration across 1000 games of Saboteur. Time moves left to right along the x-axis, from the beginning to the end of the game. The left-hand plot shows games won by the gold-diggers, and the right-hand plot those won by the saboteurs. There are two distinct groups of lines in each plot because some saboteur configurations are more likely than others (configurations with two saboteurs are twice as likely as those with only one) and the initial belief distribution reflects this.

7.6 Summary

In this chapter is has been demonstrated that the opponent model implicit in ISMCTS can be re-used for inference. To address non-locality in ISMCTS, several methods for injecting self-determinizations were tested and it was shown that bluffing can be made to emerge from ISMCTS through the use of self-determinizations. This was very effective in the game The Resistance, where particle filter inference was able to significantly improve the ability of ISMCTS to determine the identity of the spies. Additionally self-determination techniques were able to significantly mitigate against opponents using inference, particularly the bluffing method which modifies move selection of ISMCTS to select a robust move that may be a also be a good bluff.

However getting bluffing behaviour to emerge from search in more complex games such as Scotland Yard and Saboteur is difficult without access to an infeasible amount of computational resources. The larger number of states and information sets in these games slows the learning rate of ISMCTS and many more iterations (or injection of domain knowledge) would be required to achieve good results. Successful bluffing takes different forms in different games. In The Resistance and Saboteur is can be desirable to deliberately mislead other players about your hidden information, but in Scotland Yard it is perhaps better to concentrate on maximising the number of possibilities which can be more easily achieved through heuristics [72]. ISMCTS could be significantly improved in these domains by including heuristics which capture these strategic ideas, as well as by sharing information between simulations to improve the learning

rate. Techniques for capturing and reusing information from simulations are considered in depth in Chapter 8.

It has been shown that as the complexity of a game increases and the number of information sets increases, the opponent model implicit in tree search becomes poorer and results in weaker inference. This problem can be addressed by reducing a game into a smaller game by grouping together information sets in a process known as abstraction. In this chapter it has been demonstrated that ISMCTS can perform inference and bluffing and this behaviour improves with more computational resources. However algorithms which approximate a Nash-equilibrium would be expected to less exploitable than ISMCTS. One of the most useful features of ISMCTS is that it often quickly finds a good strategy, However the results in this chapter indicate that when bluffing and inference are strategically important, opponents information sets must be searched which can hinder that ability of ISMCTS to quickly find a good strategy. A subject for future work is to compare the performance and scaling of ISMCTS to other techniques which converge to a Nash-equilibrium (see Chapter 3.3.4) (particularly online MCCFR [121] and Smooth UCT [84]) for complex games where bluffing and inference are important.

Chapter 8

Information Capture and Reuse (ICARUS)

Chapters 5 and 6 introduced the ISMCTS algorithm, designed to handle different sources of hidden information and uncertainty. A large body of work exists which focuses on improving MCTS by extracting extra information from simulations, such as the RAVE enhancement which is key to the success of MCTS in Go and other domains. Enhancements to MCTS are often an effective way of improving the rate at which an MCTS player improves with increased iterations. This is discussed in detail in Chapter 3.2.3 (page 26). In particular many of these enhancements can be described as *general purpose* enhancements, which can improve the performance of MCTS without requiring any specific domain knowledge. General purpose enhancements work using two distinct processes; information *capture* (from MCTS simulations) and then *reuse* (to improve the selection and simulation policies of MCTS). There already exists many different approaches to both information capture and reuse. Increasing the learning rate of MCTS is particularly important in many complex games of imperfect information, for example in Chapter 7 it was shown that searching opponent information sets is necessary when inference and bluffing is important, but increases the amount of MCTS iterations needed to produce strong play.

Without experimentation it is difficult to determine which combination of enhancements will work best in a particular domain and there may be many valid ways of integrating a particular enhancement with a particular MCTS based algorithm (especially when multiple enhancements are combined). In this chapter a framework for *information capture and reuse* (ICARUS) is introduced, which enables the integration of existing and new enhancements of MCTS and ISMCTS to be easily defined. By separating the information capture and reuse aspects of existing enhancements and their integration into MCTS, the ICARUS framework is used to evaluate the effectiveness of a large number of permutation of these elements. In particular, the effectiveness of an enhancement depends on both the information capture method information reuse method. By separating

the two, it is possible to find the best type of information to capture and the most effective information reuse method for that information for a particular domain. This is used to find new variants of MCTS enhancements which are more effective for particular domains.

8.1 Motivation

Generally speaking, MCTS algorithms heuristically build an asymmetric partial search tree by applying machine learning, using the weak reward signal given by randomly simulating a playout to the end of the game from nodes representing intermediate positions. The tree is descended by recursively applying a multi-armed bandit formula (such as UCB1 [39]) to each tree node's counts of simulation wins and visits.

While MCTS has provided effective and even state-of-the-art decision-making in its "vanilla" form (particularly UCT [2]), it is often enhanced (for examples see [1]). Some of these enhancements incorporate external knowledge into the search, whereas others are *general purpose* enhancements which can be applied to any domain without specific knowledge. In some cases these enhancements are crucial aspects of successful MCTS programs, for example the RAVE enhancement [54] used in champion Go [5] and Hex [21] programs. In vanilla MCTS, the only information retained from a playout is the terminal reward, and the only use for that information is to update the nodes visited during the playout. Many enhancements aim to extract more data from each playout and spread the influence of that data across more of the search tree, thus increasing the value of each playout.

This chapter is concerned with the use of general purpose enhancements to improve the performance of MCTS. In some games a move that is good in one state may be good in other similar states, and general purpose MCTS enhancements can improve the performance of the algorithm by exploiting opportunities for learning in these situations. In other words, these enhancements bootstrap the learning of whether states and actions are good or bad by using analogy with similar states and actions elsewhere in the search tree. A substantial contribution of this work is the development of the ICARUS framework which formalises the correlation between states and actions, and the effects that this has on the tree and default policies of MCTS. Furthermore, the ICARUS framework is used develop and empirically investigate combination operators for MCTS enhancements, and it is show and ICARUS and it's operators to understand, categorise and invent new enhancements. Hence the effectiveness of MCTS enhancements can be explained by understanding how information is shared between states and actions and how this information is used to improve the MCTS selection and simulation policies.

The idea of enhancing an algorithm to better capture and reuse information as it executes is used in a number of search and learning algorithms. The efficiency of the α–β pruning strategy in minimax search is largely dependent on the order in which actions are visited in the tree [122]. Enhancements such

as the killer heuristic [123], history heuristic [124] and iterative deepening [125] use information gathered during the search to refine this ordering as the search progresses. Even α–β pruning itself can be seen as an information reuse enhancement, as it uses information gathered in one part of the tree to influence the search in other parts (specifically, to prune other parts entirely). Machine learning algorithms can also bootstrap learning through reuse. In *transfer learning* [126] or *lifelong learning* [127], the learner uses information learned from previous problems to bootstrap learning for the present problem. In *multitask learning* [128], the system learns to solve several problems in parallel. In both cases the system can be thought of as "learning to learn", thus these approaches are often termed *meta-learning* [129]. Typically meta-learning systems work by learning reusable features or representations, or by adjusting the parameters of an underlying learning algorithm. Although the actual methods used are different, the idea of a learning system acquiring knowledge over its lifetime as it is confronted by different problems is similar to the idea of a tree search algorithm transferring knowledge from one part of the game tree to another over the "lifetime" of a single search.

Most general purpose MCTS enhancements derive knowledge by comparing and combining simulations from different states. In this chapter it is shown that these general purpose enhancements do not always work and are sometimes detrimental to the performance of MCTS (particularly the RAVE enhancement), adding to existing observations that certain enhancements which are effective in some domains fail to provide any benefit in other domains (e.g. [130, 15]). The most effective enhancements correctly identify which states have correlated action values. This suggests that even if a general purpose enhancement is knowledge-free, there is implicit knowledge contained in the AI designer's decision of whether or not to use that enhancement.

As well as informing the choice between existing enhancements, consideration of correlated states allows entirely new enhancements to be designed. This chapter presents a new enhancement, *EPisodic Information Capture and reuse (EPIC)*, that was designed by considering correlation between states in the card game Dou Di Zhu. Dou Di Zhu has an episodic structure, where a game consists of a sequence of somewhat independent rounds, and EPIC is designed to correlate states in analogous positions within different episodes. Many games have an episodic structure, and it is demonstrated that EPIC is an effective general purpose enhancement for other games.

Capturing information in the correct way is important, but reusing it in the correct way is equally crucial. The ICARUS framework separates reuse from capture, enableing the effectiveness of different information reuse techniques to be studied. Experiments in this chapter show that the precise information reuse method has an impact on the performance of an enhancement, and in particular that the UCB1 algorithm for multi-armed bandits [39] produces strong simulation policies. This leads to an elegant MCTS algorithm which uses a bandit algorithm to select all moves in the playout, where in the MCTS tree the action value estimates correspond to information about a single state and in simulations the action value estimates correspond to information reused between many

states. Thus the only difference between the "in tree" (selection) and "out of tree" (simulation) modes of MCTS is whether the context in which the bandit algorithm executes is specific to a single state or general across a larger collection of states.

The structure of this chapter is as follows. Section 8.2 introduces in detail the idea of information capture and reuse strategies and the ICARUS framework is defined along with example ICARUS specifications for several well known MCTS enhancements. Additionally combination operators for ICARUSes are defined and criteria are established for preserving the convergence properties of the UCT algorithm in perfect information games when using ICARUS enhancements. Section 8.3 introduces a new class of enhancements (EPIC) which exploits the episodic structure of games such as Dou Di Zhu. In Section 8.4 it is shown how the ICARUS framework can be used to compare enhancements. Section 8.5 details experiments run to test different enhancements. Firstly an experiment is performed to test the effectiveness of all possible combinations of three enhancements (RAVE, MAST and EPIC) across a wide variety of games. Next experiments are presented which investigate the best selection policy for simulation enhancements and the most effective N-gram lengths for learning simulation policies. Finally results are presented on the effectiveness of ICARUS combinations with respect to computation time. Section 8.6 summarizes the results of this chapter and suggests directions for further work.

8.2 Information Capture And ReUse Strategies (ICARUSes)

An *Information Capture And ReUse Strategy (ICARUS)* is an enhancement to MCTS that collects information from visits to one area of the game tree and uses that information to inform the future play-out policy in other areas. The ICARUS framework enables the definition and analysis of such enhancements and their combinations in an instructive, formal and consistent way. Furthermore, the framework is generic enough to be able to express any kind of information reuse enhancement (for example consulting an oracle of arbitrary complexity is permitted), but imposes a structure on how information is captured and used. This allows the structure of different enhancements to be easily compared, and provides useful pointers towards the design of future enhancements.

8.2.1 Defining ICARUSes

The sharing of information between different parts of the tree is facilitated by *records*. These can be any objects. During the search, each record has a piece of *information* associated with it. The piece of information can also be any object; for example, it may be a tuple of numbers representing rewards and visit counts. The ICARUS defines three functions: the *policy function* specifying how the information is used during each MCTS playout, the *capture function* specifying which records are to be updated in response to the playout,

and the *backpropagation function* specifying how each record's information is updated. This is similar to reinforcement learning, where the policy function is to be optimised, playouts provide a performance measure and the capture and backpropagation functions define a learning mechanism. Depending on the enhancement, records can be updated for different reasons: for example some records may be updated because they were selected, and others because they were available for selection but not actually selected. *Capture contexts* are used to communicate this between the capture function and the backpropagation function.

Definition 12. Given a game as defined in Chapter 2, an *information capture and reuse strategy (ICARUS)* is a 7-tuple $(R, \Theta, \theta_{\text{initial}}, \alpha, \Psi, \xi, \omega)$ where

1. R is a nonempty set of *records*. The elements of R can be any objects.
2. Θ is a nonempty set, the *information domain*. The elements of Θ can be any objects.
3. $\theta_{\text{initial}} : R \to \Theta$ is the *initial information function*, which maps each record to a piece of information.
4. $\alpha : M^* \times (R \to \Theta) \times 2^A \to (A \to [0, 1])$ is the *policy function*. This function takes three arguments (the current move history, the current mapping of records to information, and the legal action set for the current state) and returns a probability distribution over the action set. The same function α is used during selection and simulation phases of the playout.
5. Ψ is a nonempty set of *capture contexts*. The elements of Ψ can be any objects, and are used to communicate contextual information between ξ and ω defined below.
6. $\xi : S \times M^* \to (R \times \Psi)^*$ is the *capture function*. This function takes two arguments (the root game state and the current move history) and maps them to a sequence of (record, capture context) pairs which are to be updated following a playout.
7. $\omega : \Theta \times \Psi \times \mathbb{R}^\kappa \to \Theta$ is the *backpropagation function*. This function takes three arguments (the current information for a record, the capture context specified by the capture function, and the reward vector from the simulation) and returns the new information for the record following a playout.

Algorithm 8 shows an ISMCTS algorithm using ICARUS to choose the best action from information set I_{root}. The algorithm begins by initialising the information associated with each record (lines 2–4); however, a practical implementation might want to initialise these values lazily as and when they are needed. Each iteration begins at the root node corresponding to the empty history (line 7), and samples a determinization (state) s_{root} from the root information set (line 8) which becomes the current state s for this iteration (line 9).

Each step of the playout uses the policy function α to choose an action a, depending on the current move history $[\boldsymbol{h}]^{\frown_{\rho(s)}}$ for the player about to act from state s, the current information mapping θ, and the set of available actions $A(s)$ (line 11). The current history $\boldsymbol{h}$ is updated by appending a, and the current state s is updated by applying a.

After the playout has reached a terminal state, the capture function is applied to the root determinization s_{root} and the terminal history $\boldsymbol{h}$ to obtain the sequence of (record, context) pairs to be updated (line 16). For each of these pairs, the backpropagation function ω is used to update the information associated with the record (line 17).

Algorithm 8 The MCTS algorithm using ICARUS. The algorithm takes an information set I_{root} as input and returns a legal action from that information set.

```
1:  function MCTS(I_root ∈ S/∼_i)
2:      // Initialisation
3:      for each record r do
4:          θ(r) = θ_initial(r)

5:      for many iterations do
6:          // Playout
7:          h ← ⟨⟩
8:          choose s_root ∈ I_root uniformly at random
9:          s ← s_root
10:         repeat
11:             choose a ∈ A(s) with probability α([h]^{⌢ρ(s)}, θ, A(s))(a)
12:             h ← h ⧺ a
13:             s ← f(s, a)
14:         until s is terminal

15:         // Backpropagation
16:         for each (r, ψ) ∈ ξ(s_root, h) do
17:             θ(r) ← ω(θ(r), ψ, μ(s))

18:     return the a ∈ A(I_root) that was selected most often from the root
```

The experimental domains in this chapter are games of both perfect and imperfect information, thus Algorithm 8 is designed to handle imperfect information using ISMCTS. Algorithm 8 is equivalent to a vanilla UCT implementation in a perfect information game as the information set I_{root} is a singleton $\{s_{\text{root}}\}$ (and line 8 can be omitted).

$$R^{\text{base}} = M^* \tag{Base-1}$$

$$\Theta^{\text{base}} = \mathbb{R}^\kappa \times \mathbb{N}_0 \times \mathbb{N}_0 \tag{Base-2}$$

$$\theta^{\text{base}}_{\text{initial}}(\boldsymbol{h}) = (\mathbf{0}, 0, 0) \tag{Base-3}$$

$$\alpha^{\text{base}}(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} v(\theta([\boldsymbol{h} +\!\!+ a]^\smile))\right] \tag{Base-4}$$

$$\text{where } v((\boldsymbol{q}, n, m)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c\sqrt{\dfrac{\log(m)}{n}} & \text{if } n > 0 \text{ and } m > 0 \\ +\infty & \text{if } n = 0 \text{ or } m = 0 \end{cases}$$

where $\boldsymbol{q}_\rho$ is the component of $\boldsymbol{q}$ corresponding to the player about to act at the end of $\boldsymbol{h}$

$$\Psi^{\text{base}} = \{\psi_{\text{avail}}, \psi_{\text{visit}}\} \tag{Base-5}$$

$$\begin{aligned} \xi^{\text{base}}(s, \langle a_1, \ldots, a_t\rangle) = {} & \langle([\langle a_1, \ldots, a_i\rangle]^\smile, \psi_{\text{visit}}) \ : \ 0 \le i \le t_e\rangle \\ & +\!\!+ \langle([\langle a_1, \ldots, a_{i-1}, a\rangle]^\smile, \psi_{\text{avail}}) \ : \ 0 < i \le t_e, \\ & \qquad a \in A(f(s, \langle a_1, \ldots, a_{i-1}\rangle)), a \neq a_i\rangle \end{aligned} \tag{Base-6}$$

where t_e is minimal such that $\theta([\langle a_1, \ldots, a_{t_e}\rangle]^\smile) = (\boldsymbol{q}, 0, m)$ for some $\boldsymbol{q}, m$, or $t_e = t$ if no such t_e exists

$$\omega^{\text{base}}((\boldsymbol{q}, n, m), \psi, \boldsymbol{\mu}) = \begin{cases} (\boldsymbol{q} + \boldsymbol{\mu}, n+1, m+1) & \text{if } \psi = \psi_{\text{visit}} \\ (\boldsymbol{q}, n, m+1) & \text{if } \psi = \psi_{\text{avail}} \end{cases} \tag{Base-7}$$

where $\boldsymbol{q}$ denotes the total reward, n denotes the number of visits and m denotes the availability count.

Specification 1: The baseline ICARUS definition

8.2.2 Baseline ICARUS definition

Specification 1 describes the baseline ICARUS definition used by an unenhanced search algorithm, defining the functions used in Algorithm 8. The resulting algorithm is equivalent to UCT [2] in the perfect information case and MO-ISMCTS with the UCB1 selection policy in the imperfect information case. The algorithm uses reward vectors and assumes that each player tries to maximise his own reward in a $\max^n$ fashion [117, 131], thus the algorithm can handle games with $\kappa > 2$ players as well as single-player and two-player games.

Each history has its own record (Base-1), and the information associated with a record is a total reward, a number of visits and an availability count (Base-2, Base-3). The policy is defined to use the subset-armed UCB1 algorithm (Base-4). During expansion all unexpanded actions have $n = 0$ and thus UCB1 value ∞, and so the policy chooses between them uniformly. Similarly during simulation, all actions have UCB1 value ∞ and so the simulation policy is uniform random. The capture function specifies that the records to be updated during backpropagation are those that were selected, and those that were available to be selected due to being compatible with the current determinization; this is restricted to the portion of the playout corresponding to selection and expansion, i.e. the first t_e actions (Base-6). These two collections of records are labelled with contexts ψ_{visit} and ψ_{avail} respectively (Base-5). Selected records have their rewards, visits and availabilities updated in the natural way: the simulation reward is added to the record's total reward, and the visit and availability counts are incremented by 1. Available records have their availability count incremented by 1, with reward and visit count remaining unchanged (Base-7).

Many ICARUSes apply different policies during selection, expansion and simulation. Let θ_n^{base} denote the visit count component of θ^{base}, i.e. $\theta_n^{\text{base}}([\boldsymbol{h}]^\smile)$ denotes the number of visits to history $[\boldsymbol{h}]^\smile$. A history $\boldsymbol{h}$ with available action set A_s is said to be

- a *selection node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^\smile) > 0$ and $\theta_n^{\text{base}}([\boldsymbol{h} + a]^\smile) > 0$ for all $a \in A_s$;
- an *expansion node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^\smile) > 0$ but $\theta_n^{\text{base}}([\boldsymbol{h} + a]^\smile) = 0$ for at least one $a \in A_s$;
- a *simulation node* if $\theta_n^{\text{base}}([\boldsymbol{h}]^\smile) = 0$.

It is important to note that when this terminology is used in the definitions of ICARUSes, it always relates to the baseline statistics and not to any information maintained by the ICARUS itself.

8.2.3 Enhancements in the ICARUS framework

This section casts some well-known information reuse enhancements from the literature into the ICARUS framework. These enhancements are introduced in Chapter 3.2.3 (page 26).

$$R = M^* \tag{AMAF-1}$$
$$\Theta = \mathbb{R}^\kappa \times \mathbb{N}_0 \tag{AMAF-2}$$
$$\theta_{\text{initial}}(\boldsymbol{h}) = (\mathbf{0}, 0) \tag{AMAF-3}$$
$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} v(\theta([\boldsymbol{h} +\!\!+ a]^\smile))\right] \tag{AMAF-4}$$
$$\text{where } v((\boldsymbol{q}, n)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c_{\text{AMAF}}\sqrt{\dfrac{\log\left(\sum_{b \in A_s} \theta_2([\boldsymbol{h} +\!\!+ b]^\smile)\right)}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$
where θ_2 denotes the component of θ in $\mathbb{N}_0$
$$\Psi = \{\psi\} \tag{AMAF-5}$$
$$\begin{aligned} \xi(s, \langle a_1, \ldots, a_t\rangle) = \langle (\langle a_1, \ldots, a_{i-1}, a_j\rangle, \psi) \;:\; & 0 \le i < j \le t, \\ & a_j \in A(f(s, \langle a_1, \ldots, a_{i-1}\rangle)) \\ & \text{and } \langle a_1, \ldots, a_i\rangle \text{ is a selection node}\rangle \end{aligned} \tag{AMAF-6}$$
$$\omega((\boldsymbol{q}, n), \psi, \boldsymbol{\mu}) = (\boldsymbol{q} + \boldsymbol{\mu}, n + 1) \tag{AMAF-7}$$

Specification 2: All moves as first (AMAF)

All moves as first (AMAF)

Specification 2 formulates AMAF in the ICARUS framework. Each history has its own record (AMAF-1), and the information associated with a record is a total reward and a number of visits (AMAF-2, AMAF-3). The policy uses a UCB1 formula based on the AMAF information (AMAF-4), here using as the number of trials the sum of visit counts for all currently available actions. The capture function specifies that the nodes to be updated are those siblings of nodes visited during tree descent that correspond to actions played later in the playout (AMAF-6). This is the key property of the AMAF algorithm. Backpropagation updates the rewards and visits in the natural way (AMAF-7), and does not require any contextual information (AMAF-5).

One well-known variant of AMAF is *rapid action value estimation (RAVE)* [54, 7], in which the influence of the AMAF value decays the more a node is visited. In Section 8.2.4 composition operators on ICARUSes are defined, and express RAVE as a composition of baseline and AMAF ICARUSes.

Move-average sampling technique (MAST)

MAST is defined in Specification 3. There is a record for each combination of an action and a player who plays that action (MAST-1). The information associated with a record is a total (scalar) reward and a visit count (MAST-2, MAST-3). The policy selects actions according to a Gibbs distribution, using the average reward calculated from the total reward and visit count (MAST-4).

$$R = A \times \{1, \ldots, \kappa\} \tag{MAST-1}$$
$$\Theta = \mathbb{R} \times \mathbb{N}_0 \tag{MAST-2}$$
$$\theta_{\text{initial}}(a, i) = (0, 0) \tag{MAST-3}$$
$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \frac{e^{v(a)/\tau}}{\sum_{b \in A_s} e^{v(b)/\tau}} \tag{MAST-4}$$
$$\text{where } v(a) = \begin{cases} \frac{q}{n} & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases} \text{ for } \theta(a, \rho_n) = (q, n)$$
$$\text{where } \rho_i = \rho(f(s, \langle a_1, \ldots, a_i \rangle))$$
$$\Psi = \{1, \ldots, \kappa\} \tag{MAST-5}$$
$$\xi(s, \langle a_1, \ldots, a_t \rangle) = \langle (a_i, \rho_i) \ : \ i = 1, \ldots, t \rangle \tag{MAST-6}$$
$$\omega((q, n), \rho, (\mu_1, \ldots, \mu_\kappa)) = (q + \mu_\rho, n + 1) \tag{MAST-7}$$

Specification 3: Move-average sampling technique (MAST)

Backpropagation updates the records associated with the actions played during the playout (MAST-6), with the player who played each action as contextual information (MAST-5). The total reward and number of visits are updated in the natural way (MAST-7). If the same (action, player) pair appears more than once in the playout, it is updated more than once during backpropagation.

This formulation of MAST applies the same policy throughout the playout, whereas [59] applies the Gibbs policy during expansion and simulation only. This behaviour can be implemented within the ICARUS framework by use of composition operators (Section 8.2.4).

In its original formulation, MAST uses a policy based on a Gibbs distribution. Tak et al [63] propose instead using an ε-greedy policy, i.e. replacing the policy function in Specification 3 with

$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \varepsilon \mathbb{U}[A_s] + (1 - \varepsilon) \mathbb{U}\left[\arg\max_{b \in A_s} v(b)\right] \tag{MAST-ε-greedy-4}$$

for a constant ε. With probability ε this policy chooses uniformly over all available actions; with probability $1 - \varepsilon$ it chooses uniformly over the actions whose average value is maximal.

Another possibility is to use a roulette wheel policy, in which the probability for each move is proportional to its average reward:

$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \frac{v(a)}{\sum_{b \in A_s} v(b)} \tag{MAST-Roulette-4}$$

Variants of MAST

Finnsson and Björnsson [104] describe a variant of MAST called *tree-only MAST (TO-MAST)*, in which only statistics for the actions played during selection and expansion (i.e. not during simulation) are updated. This can be defined by modifying the capture function of Specification 3:

$$\xi(s, \langle a_1, \ldots, a_t \rangle) = \langle (a_i, \rho_i) \ : \ i = 1, \ldots, t \quad \text{and } \langle a_1, \ldots, a_i \rangle \text{ is a selection or expansion node} \rangle \qquad \text{(TO-MAST-6)}$$

Finnsson and Björnsson [104] describe two refinements of MAST to enable embedding of domain specific knowledge. In *predicate-average sampling technique (PAST)*, states are labelled using a list of predicates; instead of maintaining average rewards for actions, rewards are maintained for (predicate, action) pairs consisting of a predicate that holds in a state and the action played from that state. PAST can be represented in the ICARUS framework by modifying Specification 3, including the predicate as an element of the record tuple and modifying the policy and capture functions to take predicates into account.

The second refinement is *features-to-action sampling technique (FAST)*. This uses the $TD(\lambda)$ temporal difference learning algorithm to learn a value function for actions, both offline before the search begins and online based on the MCTS playouts. In the ICARUS framework, the values learned offline can be encoded in the initial information function θ_{initial}, and the online learning by embedding $TD(\lambda)$ in the backpropagation function ω.

Last good reply (LGR)

Specification 4 gives LGR as an ICARUS. Each record specifies a move to be replied to, and the player making the reply (LGR-1). The information associated with a record is the last good action played in reply to that move by that player, or $\perp \notin A$ if no reply has yet been recorded (LGR-2, LGR-3). The policy examines the most recent move $[a_t]^{\frown_{\rho_t}}$ from the point of view of the player about to act ρ_t. If a reply has been recorded, and that reply is compatible with the current determinization, then it is played. Otherwise, a legal action is chosen uniformly at random (LGR-4). During backpropagation, the records updated are those corresponding to the actions in the playout, each action observed from the point of view of the player immediately following it (LGR-6). The context specifies the action with which that player replied, as well as the identity of the player (LGR-5). If the player won the simulated game (i.e. achieved a reward greater than zero), the action is recorded as the last good reply; if not, the existing information is retained (LGR-7).

Drake [60] only uses the reply information during simulation, whereas Specification 4 has it used for the entire playout. This is likely to be very weak. However using this definition means that the stage at which the reply information is used can be specified naturally by composition operators (Section 8.2.4) rather than as a part of the ICARUS itself.

$$R = M \times \{1, \ldots, \kappa\} \tag{LGR-1}$$
$$\Theta = A \cup \{\bot\} \tag{LGR-2}$$
$$\theta_{\text{initial}}(m, i) = \bot \tag{LGR-3}$$
$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \begin{cases} \mathbb{U}[A_s] & \text{if } \theta([a_t]^{\smile \rho_t}, \rho_t) = \bot \\ & \text{or } \theta([a_t]^{\smile \rho_t}, \rho_t) \notin A_s \\ 1 & \text{if } \theta([a_t]^{\smile \rho_t}, \rho_t) = a \\ 0 & \text{otherwise} \end{cases} \tag{LGR-4}$$
$$\text{where } \rho_i = \rho(f(s, \langle a_1, \ldots, a_i \rangle))$$
$$\Psi = A \times \{1, \ldots, \kappa\} \tag{LGR-5}$$
$$\xi(s, \langle a_1, \ldots, a_t \rangle) = \langle (([a_i]^{\smile \rho_i}, \rho_i), (a_{i+1}, \rho_i)) \ : \ i = 1, \ldots, t-1 \rangle \tag{LGR-6}$$
$$\omega(a_{\text{old}}, (a_{\text{new}}, \rho), \boldsymbol{\mu}) = \begin{cases} a_{\text{new}} & \text{if } \boldsymbol{\mu}_\rho > 0 \\ a_{\text{old}} & \text{if } \boldsymbol{\mu}_\rho \leq 0 \end{cases} \tag{LGR-7}$$

Specification 4: Last good reply (LGR)

Baier and Drake [61] describe a variant of LGR called *last good reply with forgetting (LGRF)*, in which replies that led to a loss are deleted from the reply table. Specification 4 can be modified to describe LGRF simply by modifying the backpropagation function:

$$\omega(a_{\text{old}}, (a_{\text{new}}, \rho), \boldsymbol{\mu}) = \begin{cases} a_{\text{new}} & \text{if } \boldsymbol{\mu}_\rho > 0 \\ \bot & \text{if } a_{\text{old}} = a_{\text{new}} \text{ and } \boldsymbol{\mu}_\rho \leq 0 \\ a_{\text{old}} & \text{otherwise} \end{cases} \tag{LGRF-7}$$

N-Gram Average Sampling Technique (NAST)

N-gram average sampling technique (NAST) [31] is a new enhancement created as part of the work presented in this thesis, which builds upon previous work by Stankiewicz et al [62] and Tak et al [63]. NAST generalises the notion of MAST: instead of learning values for single moves, NAST learns values for sequences of consecutive moves (indeed, MAST can be thought of as the $N = 1$ case for NAST). Experiments published in [32] and presented in Section 8.5.4 showed that NAST is effective in three imperfect information games with $n = 2$ typically giving the strongest performance.

In the context of games, an *N-gram* is a sequence of N consecutive actions. In the same way that MAST maintains average reward statistics for actions, *N-gram-Average Sampling Technique (NAST)* is introduced to maintain average reward statistics for N-grams. A 1-gram is simply an action, so MAST is a special case of NAST with $N = 1$. NAST can also be thought of as a generalisation of the last good reply principle [60]: the average reward for a 2-gram

$$R = M^n \times \{1, \ldots, \kappa\} \tag{NAST-1}$$

$$\Theta = \mathbb{R} \times \mathbb{N}_0 \tag{NAST-2}$$

$$\theta_{\text{initial}}(m_1, \ldots, m_n, i) = (0, 0) \tag{NAST-3}$$

$$\alpha(\langle a_1, \ldots, a_t \rangle, \theta, A_s)(a) = \mathbb{U}\left[\arg\max_{a \in A_s} v(\theta(\langle a_{t-n+2}, \ldots, a_t, a \rangle^{\frown}, \rho_t))\right] \tag{NAST-4}$$

$$\text{where } \rho_i = \rho(f(s, \langle a_1, \ldots, a_i \rangle))$$

$$\text{and } v((q, n)) = \begin{cases} \frac{q}{n} + c_{\text{NAST}} \sqrt{\frac{\log \Sigma}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$

$$\text{where } \Sigma = \sum_{b \in A_s} \theta_2(\langle a_{t-n+2}, \ldots, a_t, b \rangle^{\frown})$$

where θ_2 denotes the component of θ in $\mathbb{N}_0$

$$\Psi = \{1, \ldots, \kappa\} \tag{NAST-5}$$

$$\xi(s, \langle a_1, \ldots, a_t \rangle) = \langle (\langle a_i, \ldots, a_{i+n-1} \rangle, \rho_{i+n-1}) \ : \ i = 1, \ldots, t - n + 1 \rangle \tag{NAST-6}$$

$$\omega((q, n), \rho, \boldsymbol{\mu}) = (q + \boldsymbol{\mu}_\rho, n + 1) \tag{NAST-7}$$

Specification 5: n-gram average sampling technique (NAST)

$\langle a_1, a_2 \rangle$ indicates whether action a_2 is a good reply to action a_1, and more generally 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 reply to the sequence of consecutive actions $\langle a_1, \ldots, a_{N-1} \rangle$. This sequence can contain actions played by several players, including the one playing action a_N.

Stankiewicz et al [62] apply N-grams of length 2 and 3 with an ε-greedy simulation policy to the game of Havannah, achieving a significant increase in playing strength. Tak et al [63] suggest an enhancement similar to NAST which uses a combination of 1-, 2- and 3-grams, and demonstrate its effectiveness in the domain of General Game Playing. There are two key differences between these approaches and NAST: first, NAST uses only a single N-gram length, allowing the effectiveness of different lengths to be measured in isolation; second, NAST allows for simulation policies other than ε-greedy to be used, including UCB1.

NAST is defined in Specification 5. Each record is an n-gram, i.e. a sequence of n moves (NAST-1). Note that n is a parameter here; Specification 5 defines a family of enhancements for $n = 1, 2, 3, \ldots$ where n is the n-gram length. The information associated with a record is the total reward and number of visits (NAST-2, NAST-3). The policy uses these to select actions according to UCB1 (NAST-4), although it would be possible to use any of the selection policies described in Section 8.2.3 for MAST. Backpropagation updates the records as-

sociated with each sequence of n moves in the playout (NAST-6), with the player who played the last move in the sequence as contextual information (NAST-5). The total reward and number of visits are updated in the natural way (NAST-7).

Note that NAST with $n = 1$ is equivalent to MAST (Section 8.2.3) with the UCB1 policy. The idea of MAST is to measure the "goodness" of each available move independent of context. NAST instead measures the "goodness" of moves in the context of the $N - 1$ moves that preceded them. NAST is expected to offer an improvement over MAST in domains where contextual information is important to success and useful statistics about chains of moves can be learnt. NAST should also be a *robust* enhancement: in games when there is no useful information for NAST to learn, the N-gram values will be approximately equal and the simulation policy will resemble random play. NAST is unlikely to bias the playouts by reinforcing incorrect values.

Other examples

The literature contains many other examples of MCTS enhancements that involve either using information from external sources or capturing and reusing information within the search. All such approaches designed to date can be represented in the ICARUS framework. AMAF, MAST and LGR were chosen as examples because they capture and reuse information in significantly different ways, whereas many enhancements are modifications of existing ones (for example the different AMAF variants described in [132]). Furthermore, these three enhancements have led to significant increases in the power of the MCTS algorithm for diverse application domains. This section briefly describes how some other enhancements from the literature can be defined within the ICARUS framework.

Chaslot et al [133] introduce *progressive bias* and *progressive unpruning*, which use a heuristic value function to bias selection and restrict expansion respectively. In the ICARUS framework this can be achieved by encoding the heuristic in the initial information function θ_{initial} and modifying the policy function α appropriately.

Nijssen and Winands [73] propose a modification of progressive bias called *progressive history*, which replaces the heuristic function with values extracted from simulations. Within the ICARUS framework this is similar to progressive bias, except that the information is updated by the backpropagation function ω instead of being initialised heuristically.

Rimmel and Teytaud [134] introduce *contextual MCTS*, which works by mapping each terminal history to several "tiles", where a tile corresponds to a pair of (not necessarily consecutive) actions played by the same player. During backpropagation the average values of tiles are updated, and these values are used to bias simulations. When contextual MCTS is encoded as an ICARUS, the tiles become records and the policy and backpropagation functions are defined in the natural way.

The *MCTS-Solver* enhancement introduced by Winands et al [135, 136] works by backpropagating game theoretic values through the tree. A termi-

$$R = R_1 \sqcup R_2 \tag{COMB-1}$$

$$\Theta = \Theta_1 \sqcup \Theta_2 \tag{COMB-2}$$

$$\theta_{\text{initial}}(r) = \begin{cases} \theta^1_{\text{initial}}(r) & \text{if } r \in R_1 \\ \theta^2_{\text{initial}}(r) & \text{if } r \in R_2 \end{cases} \tag{COMB-3}$$

$$\alpha(\boldsymbol{h}, \theta, A_s) = \begin{cases} \alpha_1(\boldsymbol{h}, \theta, A_s) & \text{if } \boldsymbol{h} \text{ is a selection or expansion node} \\ \alpha_2(\boldsymbol{h}, \theta, A_s) & \text{if } \boldsymbol{h} \text{ is a simulation node.} \end{cases} \tag{COMB$^{\triangleright}$-4}$$

$$\Psi = \Psi_1 \sqcup \Psi_2 \tag{COMB-5}$$

$$\xi(s, \boldsymbol{h}) = \xi_1(s, \boldsymbol{h}) + \xi_2(s, \boldsymbol{h}) \tag{COMB-6}$$

$$\omega(\theta, \psi, \boldsymbol{\mu}) = \begin{cases} \omega_1(\theta, \psi, \boldsymbol{\mu}) & \text{if } \theta \in \Theta_1 \\ \omega_2(\theta, \psi, \boldsymbol{\mu}) & \text{if } \theta \in \Theta_2 \end{cases} \tag{COMB-7}$$

Specification 6: Sequential composition ($\triangleright$)

nal state is always known to be a win or a loss; at a decision node for player p, if one of the children is a known win then the node itself is a known win; if all of the children are known losses then the node itself is a known loss. This can be implemented by allowing nodes to take reward values of $+\infty$ and $-\infty$ to represent known wins and losses respectively, and modifying backpropagation to handle these values appropriately.

8.2.4 Combining ICARUSes

For a particular domain, the most effective information reuse approach is often a combination of other approaches. Thus it is useful to have well-defined ways to combine ICARUSes.

Three ways of combining ICARUSes are considered. The first is *sequential combination*. For two ICARUSes $I_1 = (R_1, \Theta_1, \theta^1_{\text{initial}}, \alpha_1, \Psi_1, \xi_1, \omega_1)$ and $I_2 = (R_2, \Theta_2, \theta^2_{\text{initial}}, \alpha_2, \Psi_2, \xi_2, \omega_2)$, the combination $I_1 \triangleright I_2$ is defined in Specification 6. Here $\sqcup$ denotes disjoint union: the sets are assumed to be disjoint, by relabelling elements if necessary. Each enhancement maintains its own records and information; the policy functions are combined so that $I_1 \triangleright I_2$ uses the policy from I_1 during selection and expansion, and the policy from I_2 during simulation. Selection and expansion nodes are defined in Section 8.2.2.

The second way of combining enhancements is *linear combination*. For two ICARUSes I_1 and I_2 as above, and a function $\lambda : \Theta^{\text{base}} \to [0, 1]$ (the mixing coefficient, which is a function of the information for the baseline ICARUS as defined in Specification 1), the combination $\lambda I_1 + (1 - \lambda) I_2$ is defined as in

Specification 6 with the exception of the policy function:

$$\alpha(\boldsymbol{h}, \theta, A_s) = \lambda\alpha_1(\boldsymbol{h}, \theta, A_s) + (1-\lambda)\alpha_2(\boldsymbol{h}, \theta, A_s) \qquad \text{(COMB}^{+}\text{-4)}$$
$$\text{where } \lambda = \lambda\left(\theta^{\text{base}}\left([\boldsymbol{h}]^{\smile}\right)\right) .$$

This can be generalized to define any convex combination of two or more enhancements in the natural way.

The third combination type is *maxilinear combination*. This is valid only for ICARUSes where the policy function has the form

$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} v(a)\right] \tag{8.1}$$

for some function $v : A \to \mathbb{R}$. For two ICARUSes I_1 and I_2 satisfying this condition with functions v_1 and v_2 respectively, and a function $\lambda : \Theta^{\text{base}} \to [0, 1]$, the combination $\lambda I_1 \oplus (1-\lambda) I_2$ is defined as in Specification 6 with the exception of the policy function:

$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} \left(\lambda v_1(a) + (1-\lambda) v_2(a)\right)\right] \qquad \text{(COMB}^{\oplus}\text{-4)}$$
$$\text{where } \lambda = \lambda\left(\theta^{\text{base}}\left([\boldsymbol{h}]^{\smile}\right)\right) .$$

For example, this allows RAVE [54] to be defined as

$$I_{\text{RAVE}} = \lambda_{\text{RAVE}} I_{\text{AMAF}} \oplus (1 - \lambda_{\text{RAVE}}) I_{\text{Baseline}} \tag{8.2}$$

where

$$\lambda_{\text{RAVE}}(\boldsymbol{q}, n, m) = \sqrt{\frac{k}{3n+k}} \tag{8.3}$$

for some constant k (which specifies the number of visits, i.e. the value of n, for which $\lambda_{\text{RAVE}} = 0.5$). Again, maxilinear combination can be generalised to combine more than two ICARUSes.

All ways of combining ICARUSes make use of information from the baseline definition (Section 8.2.2) in some way, whether to determine the current stage (selection, expansion or simulation) of the playout or to vary the combination coefficient. Thus for a combination to make sense, it must incorporate the baseline ICARUS.

8.2.5 Convergence properties of ICARUSes

Kocsis and Szepesvári [2] prove that, for games of perfect information, UCT converges on the optimal move in the limit. That is, as the number of iterations tends to infinity, the probability of selecting a suboptimal move tends to zero.

Definition 13. Consider a history $\boldsymbol{h}$, which when applied to the initial game state s_0 gives a state $f(s_0, \boldsymbol{h}) = s$ with legal actions A_s. Let $A_s^* \subseteq A_s$ be the

set of optimal actions from state s. An ICARUS I with policy α is *convergent* if, for all $a \in A_s \setminus A_s^*$,

$$\lim_{\text{iterations}\to\infty} \alpha(\boldsymbol{h}, \theta, A_s)(a) \to 0\,. \tag{8.4}$$

That is, for every suboptimal action a, the probability assigned to a by the playout policy tends to zero in the limit.

For the baseline ICARUS (Specification 1) applied to a game of perfect information:

Lemma 1. The baseline ICARUS is convergent.

Proof. It follows immediately from [2, Theorem 5] that (8.4) holds for $\alpha = \alpha^{\text{base}}$. □

Lemma 2. There exists an iteration number t such that, after t iterations, $\boldsymbol{h}$ is a selection node.

Proof. From [2, Theorem 3], there exists a constant k such that, after t iterations, the number of visits to $\boldsymbol{h}$ is at least $\lceil k \log t \rceil$. In particular there is a t such that $\lceil k \log t \rceil \geq 2$, which implies that $\boldsymbol{h}$ is expanded and is now a selection node. □

From these results, it can be shown that certain combinations of ICARUS are convergent:

Theorem 1. Let I_1 and I_2 be ICARUSes such that I_1 is convergent. Let $\lambda : \Theta^{\text{base}} \to [0,1]$ such that $\lambda(\boldsymbol{q}, n, m) \to 0$ as $n \to \infty$. Then the following ICARUSes are convergent:

(i) $\lambda I_2 + (1-\lambda) I_1$;

(ii) $\lambda I_2 \oplus (1-\lambda) I_1$ (if defined);

(iii) $I_1 \triangleright I_2$.

Proof. The convergence of (i) and (ii) follows from the fact that λ tends to 0 as the number of visits to a node tends to infinity. This ensures that I_1 dominates in the limit, so the combination inherits its convergent behaviour.

The convergence of (iii) follows from Lemma 2: after some finite number of iterations, all nodes are selection nodes (recall Chapter 2 that "games" are defined to have a finite number of states). At this point, $I_1 \triangleright I_2$ behaves identically to I_1 and thus converges. □

It follows from Lemma 1 and Theorem 1 (ii) that RAVE (8.2) converges. The ICARUS combinations used in the experiments in Section 8.5 (Table 8.1) all have the form $I_1 \triangleright I_2$ for $I_1 \in \{I_{\text{Baseline}}, I_{\text{RAVE}}\}$, and so also converge.

Note that these convergence results only apply to games of perfect information. For games of imperfect information, no proof equivalent to that of Kocsis

and Szepesvári [2] can be presented for ISMCTS. Indeed, in experiments which are not detailed here it was observed that ISMCTS does not converge in the sense of Definition 13, either oscillating between several policies or settling on a policy which does not form part of a Nash equilibrium. Understanding the behaviour of ISMCTS for large number of iterations is an open problem. However if convergence to a Nash equilibrium is important then other techniques such as counterfactual regret minimization (introduced in Chapter 3.3.4 page 32) are more appropriate, but may not converge to a good strategy as quickly as ISMCTS. Nevertheless, designing enhancements that converge in the perfect information case seems to be a useful way to obtain plausible play across all domains.

8.3 Episodic Information Capture and Reuse

This section introduces *EPisodic Information Capture and reuse (EPIC)*, an enhancement designed within the ICARUS framework. The unique feature of EPIC is how information is captured, i.e. which states are considered to be correlated. A game can be be divided into a number of time windows called *episodes*, and share information between states that correspond to the same position in different episodes. That is, states reached by the same sequence of actions from the beginning of their respective episodes, but where the starting points of those episodes may be different. The aim of information capture and reuse is to exploit the correlations between the values of nodes in different parts of the game tree. EPIC is designed to exploit the correlation between *subtrees* rather than individual nodes.

Many games are episodic in nature: multiplayer games have a sequence of opponents' turns; ladder games such as Dou Di Zhu [26], President and Cheat [137] have a sequence of moves until a "reset" action occurs; strategic board and card games such as Lord Of The Rings: The Confrontation [27] and Magic: The Gathering [52] have compound turns consisting of several individual decisions. If the episodes truly are independent, this implies that the strength of a policy for a particular episode does not depend on the context of where that episode occurs in the game. Thus strong play overall can be achieved by constructing a good policy for each episode, and combining these policies to obtain a policy for the full game. The fact that the same episode occurs in several different parts of the game tree implies that a naïve tree search algorithm must rediscover the strong episode policy many times. EPIC aims to discover the episode policy only once, and reapply it throughout the game tree.

The assumption that episodes are independent of context may be reasonable but is never strictly true in real games. In experiments, EPIC is combined with the baseline player, with EPIC used only as a simulation policy. This ensures that the baseline tree policy can tailor itself to the context of the current episode if that context matters, whilst the simulation policy that uses episode information but ignores context is still likely to be much stronger than a random policy.

The idea of episodes is not specific to any particular game, but it is also not universal. Games such as Chess and Go do not have a natural episodic structure, or rather the highly spatial nature of these games means that a purely temporal notion of episode does not make sense. However, even for these games, notions such as combinations in Chess [138] and joseki or tesuji in Go [139, 140] are a type of spatial episode. This work only considered temporal episodes, consisting of consecutive game turns. Nevertheless, a spatially episodic nature could conceivably be exploited by enhancements similar to EPIC.

8.3.1 Definition of EPIC

Although episodes do not feature in the formal definition of a game given in Chapter 2 the division into episodes is usually highly intuitive when it exists, for example identifying the start of a new round or hand in a card game or a particular event happening such as a piece being captured. In other words, it is a piece of domain knowledge which does not require expert insight into the game. In most cases the episode information can be read directly from an implementation's representation of the game state.

Let E be a finite nonempty set of *episode labels*. Define $e : S \to E \cup \{//\}$, the *episode function*. The element $// \notin E$ is the *continuation label*.

Consider a game Γ. An *episode* of Γ is a subtree of Γ's game tree such that

1. for the subtree's root node s_r, $e(s_r) \in E$;
2. for all leaf nodes s_l of the subtree, either $e(s_l) \in E$ or s_l is a terminal state;
3. for all other nodes s, $e(s) = //$.

If the initial state has episode label in E then the episodes partition the game tree, as illustrated in Figure 8.1.

The *position-in-episode* of a history $\langle a_1, \ldots, a_t \rangle \in A^*$ is the pair

$$\pi(\langle a_1, \ldots, a_t \rangle) = (e(s_i), \langle a_{i+1}, \ldots, a_t \rangle) \tag{8.5}$$

where $s_i = f(s_0, \langle a_1, \ldots, a_i \rangle)$ and i is maximal such that $e(s_i) \neq //$. The position-in-episode specifies the label of the current episode and the suffix of the history restricted to that episode. See Figure 8.1 for an example. The position-in-episode of a move history is defined similarly. Positions-in-episode are always defined relative to the initial state s_0 of the game, regardless of the current state.

Conceptually an episode should be somewhat independent of its context: when episodes with the same root label appear in different parts of the game tree, they should look similar and have similar outcomes for all players. In other words, if two different histories have the same positions-in-episode, they should have similar rewards. This statement is deliberately vague: the assignment of episode labels to states is a decision to be made when designing the AI agent

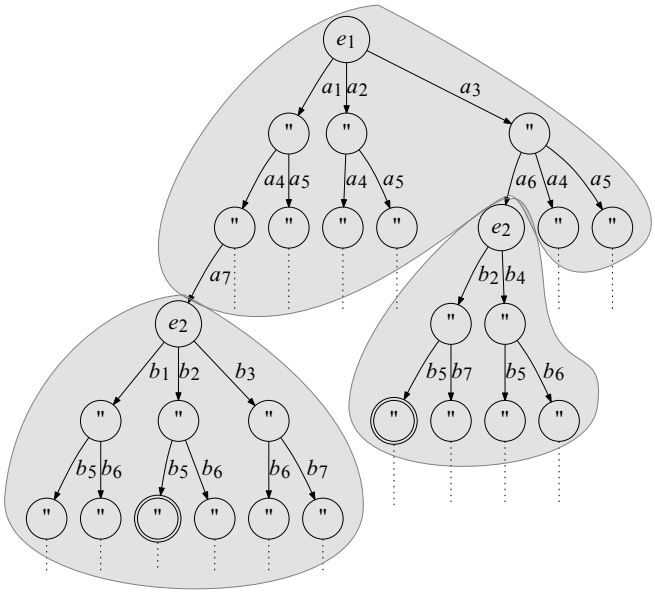

Figure 8.1: Example of a game tree where each node is labelled with its episode label. The shaded regions show the partitioning of the tree into episodes. The circled nodes have the same position-in-episode, namely $(e_2, \langle b_2, b_5 \rangle)$.

$$R = E \times M^* \tag{EPIC-1}$$

$$\Theta = \mathbb{R}^\kappa \times \mathbb{N}_0 \times \mathbb{N}_0 \tag{EPIC-2}$$

$$\theta_{\text{initial}}(e, \boldsymbol{h}) = (\mathbf{0}, 0, 0) \tag{EPIC-3}$$

$$\alpha(\boldsymbol{h}, \theta, A_s) = \mathbb{U}\left[\arg\max_{a \in A_s} v(\theta(\pi([\boldsymbol{h} +\!\!+ a]^\smile)))\right] \tag{EPIC-4}$$

$$\text{where } v((\boldsymbol{q}, n, m)) = \begin{cases} \dfrac{\boldsymbol{q}_\rho}{n} + c_{\text{EPIC}}\sqrt{\dfrac{\log m}{n}} & \text{if } n > 0 \\ +\infty & \text{if } n = 0 \end{cases}$$

$$\Psi = \{\psi_{\text{avail}}, \psi_{\text{visit}}\} \tag{EPIC-5}$$

$$\begin{aligned} \xi(s, \langle a_1, \ldots, a_t\rangle) = {} & \langle(\pi([\langle a_1, \ldots, a_i\rangle]^\smile), \psi_{\text{visit}}) \ : \ 0 \le i \le t\rangle \\ & +\!\!+ \langle(\pi([\langle a_1, \ldots, a_{i-1}, a\rangle]^\smile), \psi_{\text{avail}}) \ : \ 0 < i \le t, \\ & \qquad a \in A(f(s, \langle a_1, \ldots, a_{i-1}\rangle)), a \ne a_i\rangle \end{aligned} \tag{EPIC-6}$$

$$\omega((\boldsymbol{q}, n, m), \psi, \boldsymbol{\mu}) = \begin{cases} (\boldsymbol{q} + \boldsymbol{\mu}, n + 1, m + 1) & \text{if } \psi = \psi_{\text{visit}} \\ (\boldsymbol{q}, n, m + 1) & \text{if } \psi = \psi_{\text{avail}} \end{cases} \tag{EPIC-7}$$

Specification 7: Episodic information capture and reuse (EPIC)

rather than a part of the formal game definition. Examples of episode functions for the experimental domains are given in Section 8.3.2.

Specification 7 defines EPIC with game specific episodes as an ICARUS. The records used by EPIC are the positions-in-episode (EPIC-1), each of which has the standard ISMCTS information of total reward, visit count and availability count (EPIC-2, EPIC-3). The positions-in-episode for a particular episode label can be organised into a tree structure. Each history is mapped to its position-in-episode. If EPIC is combined with the baseline algorithm using sequential combination then during simulation, subset-armed UCB1 selection is applied according to the current position-in-episode (EPIC-4): effectively this means that the simulation policy for the overall search is provided by the tree policy in the episode trees. Rewards are backpropagated as in the baseline case, but in the episode trees rather than the full tree (EPIC-5, EPIC-6, EPIC-7).

8.3.2 Episode functions for experimental domains

In this section the EPIC episode structure is defined for each test domain.

Dou Di Zhu

The EPIC episode function for Dou Di Zhu is defined as follows. Set $E = \{L_1, L_2, L_3\}$ and

$$e(s) = \begin{cases} L_i & \text{if player } i \text{ makes a leading play from state } s \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.6}$$

Here an episode is a stream from one leading play to the next. EPIC aims to improve on a weakness of ISMCTS for Dou Di Zhu caused by the high branching factor associated with leading plays (several hundred in some cases [26]). EPIC helps to more accurately model the sequence of plays that follows each leading play, and thus give a more accurate evaluation of each leading play, without having to rediscover this sequence in several parts of the tree.

Hearts

To define the episode function for EPIC, set $E = \{D, L_1, L_2, L_3, L_4\}$ and

$$e(s) = \begin{cases} D & \text{if } s \text{ is the chance state for the deal in a new round} \\ L_i & \text{if player } i \text{ begins a trick from state } s \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.7}$$

The first episode in a game of Hearts encompasses the dealing and card passing stages; subsequent episodes are single tricks. Here EPIC makes use of the fact that similar tricks may appear in many different places in the search tree.

Lord of the Rings: The Confrontation

For EPIC's episode function, set $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if there is no combat in progress and } \rho(s) = i \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.8}$$

An episode begins when a player moves a piece. If this movement does not result in combat, the episode ends immediately. Otherwise, the episode continues until combat is resolved. The benefit of EPIC here is twofold: it collects statistics for each movement action in a manner similar to MAST, and it refines the simulation policy for combat.

Checkers

To apply EPIC to Checkers, set $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if a non-chained capture move is available, and } \rho(s) = i \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.9}$$

Here a "non-chained" capture is one which does not continue a chain of captures, but may start one. An episode begins when a capture is made, and ends the next time a capture is made (by either player) on a subsequent turn. The episodes here are intended to exploit the notion that the value of moves changes significantly when a capture is made.

Othello

The EPIC episode function is defined by $E = \{M_1, M_2\}$ and

$$e(s) = \begin{cases} M_i & \text{if the previous move was on the edge of the board, and } \rho(s) = i \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.10}$$

An episode begins on the turn after a player places a new piece on one of the 28 squares around the edge of the board. This captures the strategic notion that controlling the edges of the board is important: pieces placed on the edge are difficult to capture but create opportunities to capture opponent pieces.

Backgammon

The compound turns of Backgammon give a natural episode structure. Set $E = \{R\}$ and

$$e(s) = \begin{cases} R & \text{if the dice are about to be rolled} \\ \prime\prime & \text{otherwise.} \end{cases} \tag{8.11}$$

Thus an episode consists of a full turn: the dice roll and the two or four moves that follow it.

8.4 Comparing ICARUSes

Having a common notation for information reuse enhancements provides a tool for their analysis and comparison. Common themes that occur in several enhancements can be identified, it is easy to see which enhancements deviate from them, and use them as building blocks to define new enhancements. For example, consider the following observations from Specifications 1–4 and 7:

- Baseline, AMAF and EPIC all have records that are, or contain, move histories (Base-1, AMAF-1, EPIC-1). This implies that their records have a tree or forest structure. In contrast, MAST and LGR have a flat record structure with one record per move or action (MAST-1, LGR-1).
- For all enhancements studied here with the exception of LGR, the information associated with a record consists of a total reward (vector or scalar), a visit count, and in some cases an availability count (Base-2, AMAF-2, MAST-2, EPIC-2).

- Baseline, AMAF and EPIC all use some variation on the UCB1 policy (Base-4, AMAF-4, EPIC-4).
- LGR is the only enhancement whose policy explicitly sets the probabilities of some moves to 1, giving a simulation policy that is more deterministic than for other enhancements (LGR-4).
- Ignoring updates for availability, most of these ICARUSes update one record per state visited in the playout (Base-6, MAST-6, LGR-6, EPIC-6). The exception is AMAF, which potentially updates several records per playout step (AMAF-6).

At a higher level, sets of nodes which share information can be identified. There are two ways in which information can be shared between nodes via records:

1. nodes use information from the same record during playout and update that record during backpropagation; or
2. nodes use information from different records but a visit to one node causes both records to be updated.

In other words, information can be shared by either writing to one record that is read by many nodes, or writing to many records that are each read by one node. MAST, LGR and EPIC are of the former type, whereas AMAF is of the latter type.

Figure 8.2 illustrates this information sharing. In each tree, information is shared between the nodes connected by a dotted line. Comparing these types of pictures with natural intuition about the game, particularly which states are expected to be correlated, gives insight into which enhancements are likely to work well for which games. Looking at Figure 8.2 and the corresponding ICARUS definitions, the following observations can be made about the type of games for which each enhancement should be effective:

- AMAF: Works well for games where the strength of an action is often independent of time. This has shown to be true in games such as Go [7] and Hex [21], where the RAVE enhancement works well. Both of these games are characterized by moves which, with a few exceptions, are fixed in place once made.
- MAST: Effective when the quality of an action is independent of the state from which it is played. This is effective in General Game Playing where little is known about the state and it is somewhat true for most games. (Arguably any interesting game that people would play will at least slightly satisfy this property, as it would otherwise be difficult for human players to develop an intuition of strategy). This is similar to AMAF, but assumes even less about the dependence of action value on context.
- LGR: Works well for games with a notion of "sente" (such as Go), where some moves imply a certain reply is necessary.

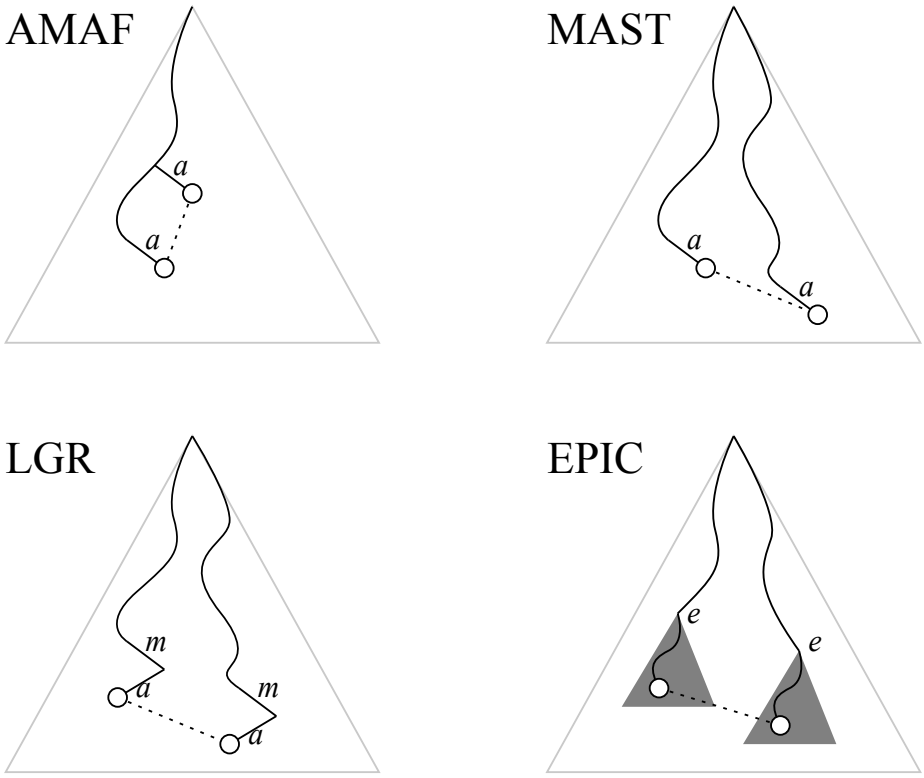

Figure 8.2: A pictorial representation of node correlation for various information capture and reuse strategies. In each case, the strategy is based on the notion that the values of the nodes connected by a dotted line are correlated, and thus sharing information between those nodes is beneficial.

- EPIC: Useful in games with an episodic nature (e.g. ladder-based card games) or complex compound turns (such as LOTR:C), where contextual information can be learned from these episodes.

The information capture methods of MAST, LGR and EPIC all belong to a class which learns the value of actions given some episode preceding the action. In the case of MAST and LGR the episodes are of fixed length (1 and 2 respectively) whereas EPIC uses episodes that are aligned to the natural episodic structure of a particular game. Fixed length episodes require no domain knowledge to implement and can be represented as *N-grams* (a sequence of N consecutive moves). N-grams have already been demonstrated as effective in capturing information with MCTS search in Havannah [62] and General Game Playing [63]. Hence as for MAST and LGR, EPIC does not absolutely require game-specific information about the nature of episodes, and may be seen as a generalisation of MAST and LGR.

There are also possibilities for other new types of enhancement that can be built with the ICARUS framework. Figure 8.2 illustrates enhancements which exploit a particular type of correlation between states in a game, but any other type of correlation can be encoded as an information capture enhancement within the ICARUS framework. Similarly there are many ways of injecting knowledge into the MCTS algorithm, with tree selection biasing and heavy playouts being amongst the most common [1, 141]. New techniques for injecting knowledge into MCTS can be combined with any existing information reuse enhancement. One potential application for the ICARUS framework could be to use techniques from machine learning (for example genetic programming) to automatically search the space of possible enhancements to discover new ones which work for a particular game. There are currently no widely used enhancements to MCTS which deal explicitly with hidden information and uncertainty in games. ICARUS provides a framework for exploring new enhancements in this area, for example considering information asymmetry when performing information capture.

8.5 Experiments

In this section experiments are presented which test the effectivness of different ICARUS combinations across a wide variety of games.

8.5.1 Strength of ICARUS combinations

A wide range of experiments were conducted to compare ICARUSes and combinations of ICARUSes for the six domains listed in Section 8.3.2, with aim to compare the performance of enhancements for games of perfect and imperfect information, and investigate whether combinations of enhancements can be greater than the sum of their parts.

In each experiment the opponent was an unenhanced player using MO-ISMCTS (for games of imperfect information) or UCT (for games of perfect information), with 5000 playouts per decision in all cases. Comparing the effectiveness of enhancements if the games were being played randomly would not allow for any useful conclusions to be made, so performance of MO-ISMCTS was evaluated for each game. Previous work [27] presented in Chapter 5.2.4 (page 83) showed the ISMCTS player was on a par with the strongest setting of a commercial Dou Di Zhu AI developed by AI Factory Ltd[1]. It was also shown in Chapter 6.3.2 (page 95) that MO-ISMCTS is on a par with intermediate-to-expert level human players for LOTR:C. For Hearts a trial of MO-ISMCTS against the AI for Hearts which ships with the Microsoft Windows 7 operating system was conducted, again showing strength parity. For Checkers, Othello and Backgammon, unenhanced UCT is significantly worse than state-of-the-art AI players, but largely because the state-of-the-art is so advanced for these games.

[1] `www.aifactory.co.uk`.

Name	EPIC	RAVE	MAST	LGRF	ICARUS specification
–					I_{Baseline}
E	•				$I_{\text{Baseline}} \triangleright I_{\text{EPIC}}$
R		•			I_{RAVE}
ER	•	•			$I_{\text{RAVE}} \triangleright I_{\text{EPIC}}$
M			•		$I_{\text{Baseline}} \triangleright I_{\text{MAST}}$
EM	•		•		$I_{\text{Baseline}} \triangleright \frac{1}{2}(I_{\text{MAST}} + I_{\text{EPIC}})$
RM		•	•		$I_{\text{RAVE}} \triangleright I_{\text{MAST}}$
ERM	•	•	•		$I_{\text{RAVE}} \triangleright \frac{1}{2}(I_{\text{MAST}} + I_{\text{EPIC}})$
L				•	$I_{\text{Baseline}} \triangleright I_{\text{LGRF}}$
EL	•			•	$I_{\text{Baseline}} \triangleright \frac{1}{2}(I_{\text{LGRF}} + I_{\text{EPIC}})$
RL		•		•	$I_{\text{RAVE}} \triangleright I_{\text{LGRF}}$
ERL	•	•		•	$I_{\text{RAVE}} \triangleright \frac{1}{2}(I_{\text{LGRF}} + I_{\text{EPIC}})$
ML			•	•	$I_{\text{Baseline}} \triangleright \frac{1}{2}(I_{\text{MAST}} + I_{\text{LGRF}})$
EML	•		•	•	$I_{\text{Baseline}} \triangleright \frac{1}{3}(I_{\text{MAST}} + I_{\text{LGRF}} + I_{\text{EPIC}})$
RML		•	•	•	$I_{\text{RAVE}} \triangleright \frac{1}{2}(I_{\text{MAST}} + I_{\text{LGRF}})$
ERML	•	•	•	•	$I_{\text{RAVE}} \triangleright \frac{1}{3}(I_{\text{MAST}} + I_{\text{LGRF}} + I_{\text{EPIC}})$

Table 8.1: A list of ICARUS combinations tested and their abbreviated names

Anecdotally, unenhanced UCT appears to be around the level of a novice human player for these games, making plausible choices and avoiding obvious mistakes.

Sixteen combinations of ICARUSes listed in Table 8.1 are compared, for the six games EPIC episodes were defined for in Section 8.3.2. The component enhancements used are:

- the baseline ICARUS (Section 8.2.2);
- RAVE (Section 8.2.4), itself based on AMAF (Section 8.2.3);
- MAST (Section 8.2.3);
- LGRF, i.e. LGR with forgetting (Section 8.2.3);
- EPIC (Section 8.3) using the episode functions for each game described in Section 8.3.2.

Where the enhancements use parameters, an initial round of parameter tuning experiments was performed to set the values for the main experiment. The parameter values are $c = 0.7$ for the exploration constant in both the baseline player and EPIC, $k = 250$ for RAVE, and $\tau = 1$ for MAST. These values give consistently good performance (relative to other parameter settings) across all games.

The ICARUSes listed in Table 8.1 cover all subsets of the four enhancements tested here. Each enhancement was originally designed for a specific phase of the MCTS iteration: RAVE [54] for selection, and MAST [59], LGR [60] and EPIC (Section 8.3) for simulation. The sequential combinations, using the $\triangleright$

operator, are defined to use each enhancement only for the appropriate phase. Where more than one enhancement is used for one of the phases, they were combined using linear combination (as described in Section 8.2.4) with equal weights. Equivalently, at each step in the simulation one of the enhancements was chosen at random and then the action chosen according to that enhancement's policy. For example if the simulation enhancement is $\frac{1}{2}(I_{\text{LGRF}} + I_{\text{EPIC}})$, the simulation plays according to LGRF with probability $\frac{1}{2}$ and according to EPIC with probability $\frac{1}{2}$. The definition (Equation 8.2) of I_{RAVE} as a maxilinear combination of I_{AMAF} and I_{Baseline} with decaying weight λ_{RAVE} is as used by Gelly and Silver [54].

For a κ-player game, one instance of the ICARUS combination in question was played against $\kappa - 1$ instances of the baseline player (unenhanced MO-ISMCTS). Each algorithm (enhanced and baseline) uses 5 000 MCTS iterations per decision. For Dou Di Zhu the ICARUS player plays as the Landlord against two non-Landlord baseline players. For Hearts the ICARUS player plays against three baseline players. For LOTR:C the ICARUS player plays as both Light and Dark against a baseline player (and due to the asymmetry of the game the results for each player are presented separately). For Checkers, Othello and Backgammon the ICARUS player plays against a baseline player, playing half the games as white and half as black. Each experiment played a large number of games (between 1000 and 2500) across a cluster of PCs, using just over one CPU-year in total. Win rates are calculated with Clopper-Pearson intervals [38] at the 95% confidence level. Each game of Dou Di Zhu, LOTR:C and Backgammon has a clear winner with no possibility of a draw. In Hearts finishing in first or equal first place is counted as a win, and any other outcome as a loss. In Checkers (where draws are common) and Othello (where draws are possible but rare), only the number of wins are counted; draws are counted as losses.

Average results for each enhancement are presented in Figure 8.3. For each enhancement, aggregate results for those combinations are presented in Table 8.1 which feature the given enhancement, and compare with the results for those combinations which do not. For example in the pairs of bars labelled "RAVE", the left-hand (diagonally shaded) bar sums the results for the eight combinations in Table 8.1 for which the RAVE column is blank, and the right-hand (solid shaded) bar for the eight combinations where the RAVE column is marked •. Where each combination in Table 8.1 was tested for 1000 trials, each of the bars in Figure 8.3 represents 8000 trials.

Results for individual combinations are shown in Figure 8.4. The dotted lines indicate 95% confidence intervals for evenly-matched UCT (perfect information) or MO-ISMCTS (imperfect information) players. Any AI with results above this band is significantly better than an unenhanced player.

Figure 8.3 shows that EPIC provides a significant improvement for all games (not quite reaching 95% significance for LOTR:C as Light). Analysis of variance (ANOVA) over all results shows that EPIC yields an improvement significant at the 99.9% level. The improvement is particularly marked for Dou Di Zhu, which has a strongly episodic nature, since during an episode (a *ladder*) many

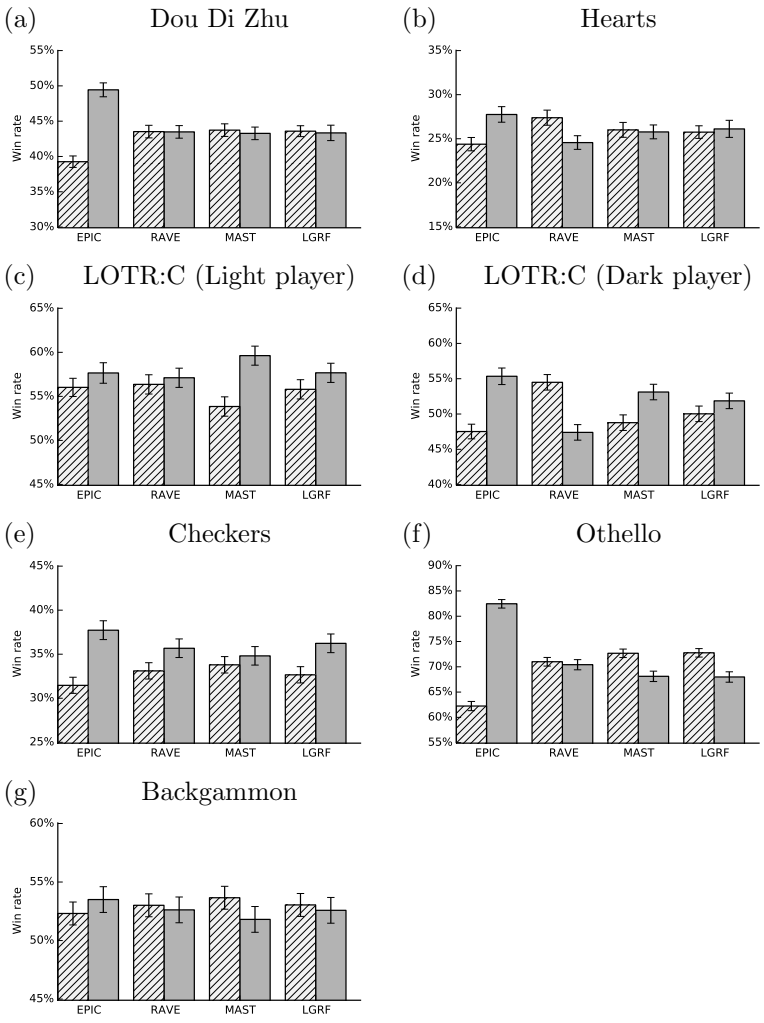

Figure 8.3: Average playing strength results for four ICARUSes tested in six different games. In each pair of bars, the left-hand bar is the average win rate over the eight combinations (out of the 16 listed in Table 8.1) not featuring the enhancement in question, and the right-hand bar the average over the combinations that do feature the enhancement.

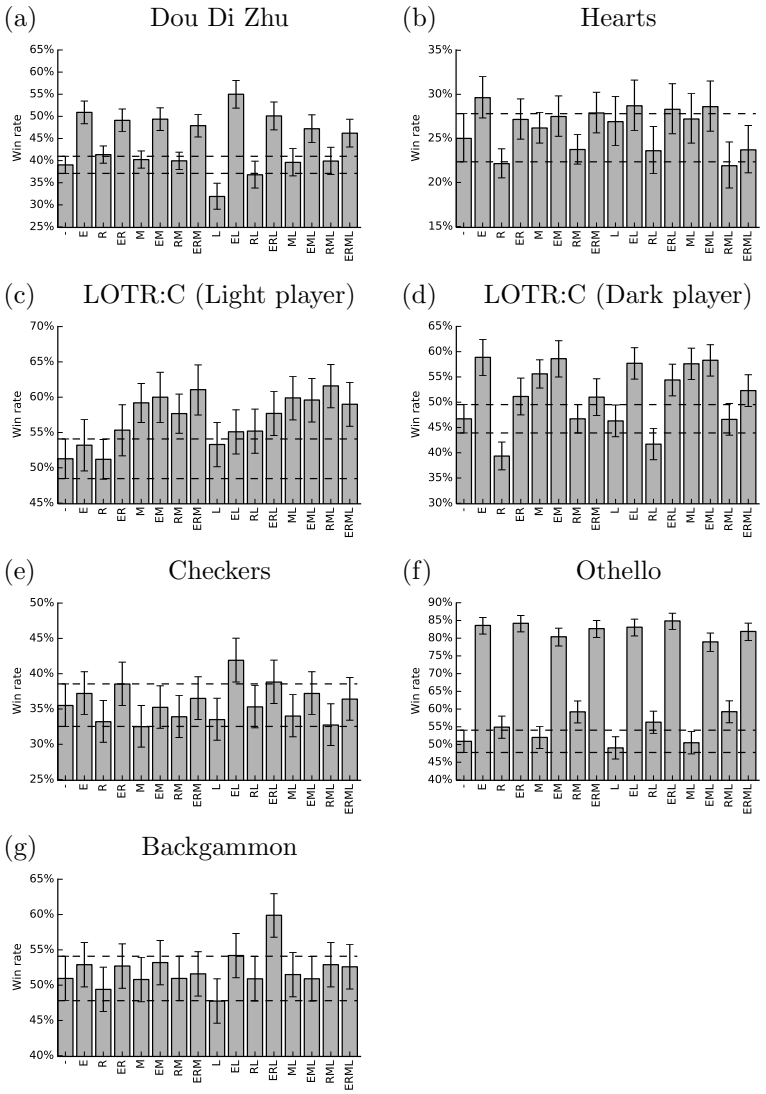

Figure 8.4: Playing strength results for ICARUS combinations (the names are listed in Table 8.1). For each ICARUS the percentage of games won is presented along with 95% confidence intervals. In each experiment 5000 MCTS iteration were used and the opponents were all baseline players (−). Dashed lines show the empirically measured 95% confidence interval for evenly-matched UCT or MO-ISMCTS players; any player with results above this baseline is significantly stronger than unenhanced UCT or MO-ISMCTS.

cards cannot be played, but these unplayable cards *will* be played in future episodes. Furthermore, the order of episodes is not often important for Dou Di Zhu. There is also some independence between tricks in Hearts and between turns in LOTR:C and Backgammon, though less than between ladders in Dou Di Zhu, so that the improvements due to EPIC, while significant, are not so marked. Checkers and Othello lack a natural episodic structure, so instead basic notions of strategically important moves (captures in Checkers and edge piece placement in Othello) are used to to delimit episodes. This works in both games, and for Othello is particularly effective.

Figure 8.3 also shows that RAVE is not effective in the imperfect information games. It is actually detrimental to performance in Hearts, and as the Dark player in LOTR:C, where the order in which moves are played is important to the strategy of the game. This makes intuitive sense: for example in Hearts the value of a particular card is very much dependent on when that card is played. For example, leading a trick with $K\spadesuit$ will be either a reasonably good or a terribly bad idea, depending whether the high-scoring $Q\spadesuit$ has already been played, so that AMAF/RAVE will fail to correctly learn whether $K\spadesuit$ is a good move. For Dou Di Zhu and LOTR:C as the Light player, RAVE did not make matters worse, but nor did it significantly improve playing strength. The only game where RAVE is significantly beneficial in experiments is Checkers.

MAST, LGRF and EPIC are more robust enhancements than RAVE: they are sometimes beneficial, and never detrimental to a statistically significant degree. As observed by Tom and Müller [130], RAVE performs poorly in situations where relative ordering of moves is important. However MAST is based on a similar principle, and is more robust. One intuition for this is that RAVE alters the tree policy in a way that can reinforce the assumption that move ordering is not important in situations where this assumption is false. MAST on the other hand learns whether a move is good or bad on average. This implies that if move ordering is not important to the goodness of a move, MAST will reinforce selection of this move, whereas if move ordering is important MAST will enforce no preference over selecting the move, falling back to the behaviour of the default random policy of MCTS. Thus the potential for MAST to learn the wrong thing, as opposed to learning nothing at all, is much less than for RAVE.

Figure 8.4 shows that linear combinations of the enhancements are often greater than the sum of their parts. LGRF produced limited improvements on its own, but it enhanced other techniques (e.g. consider EL beat both E and L for Dou Di Zhu, despite L being significantly worse than −, and ML beat both M and L for LOTR:C as Dark despite no significant difference between − and L). MAST was generally effective across all games, but proved even more effective in combination (e.g. consider RML beat all of R, M and L for LOTR:C as Light and ML beat both M and L for LOTR:C as Dark). EPIC performed strongly across all games, but was most effective in combination for Dou Di Zhu (EL beat both E and L) and LOTR:C as Light (ERM beat all of E, R and M). From Figure 8.4 (g) it appears that ERL is a particularly strong combination for Backgammon, yet analysis of variance (ANOVA) does not show this to be statistically significant.

Analysis of the MCTS trees shows that the final number of visits for the chosen action is significantly (often 10–30%) higher for EPIC than for the baseline and LGRF, RAVE or MAST. Hence EPIC is converging more quickly to an action which is generally stronger. The average reward assigned by EPIC to the chosen action is also markedly different from that in trees using other enhancements, and presumably this value is more accurate given the increased playing strength due to EPIC. Of the enhancements tested here, only RAVE has a significant impact on the depth of the MCTS tree. RAVE tends to construct deeper trees, with the deepest node being on average one ply deeper than for the other algorithms. That RAVE is detrimental despite this suggests that it is expanding too deeply the "wrong" areas of the tree, although the fact that the degree of exploitation at the root is the same as without RAVE suggests that this mostly occurs at levels in the tree below the root.

8.5.2 EPIC compared to NAST

EPIC requires a game-specific episode function. To determine whether this domain knowledge is strictly necessary, EPIC was tested against NAST with $n = 2$. Both enhancements use UCB1 as a simulation policy, the difference being the context in which the multi-armed bandit statistics are collected: for EPIC the context is game-specific, whereas for $n = 2$ NAST the context is the previous move in the game (similar to LGRF). Results are shown in Figure 8.5. In Othello, it was found that NAST is significantly better than EPIC; in all other cases, NAST and EPIC have the same performance within 95% significance. As a corollary to this, it can be concluded that NAST is at least as robust as EPIC for the games studied.

EPIC remains an interesting method for injecting knowledge into search in other game domains, but for these games it is clear that the episodes do not need to be so carefully chosen. Both EPIC and NAST are methods for learning useful lines of play for the playout policy; EPIC achieves this by learning fragments of lines delimited by game-specific episodes, whereas NAST with $N = 2$ essentially learns a Markov model. For the games tested here the Markov model is rich enough to capture useful information, but for other games a more sophisticated model such as EPIC may be required. The experiment presented in Section 8.5.1 was repeated with NAST instead of EPIC, but the results were not significantly different (in terms of which combinations were and were not effective).

8.5.3 Simulation policies for MAST

This experiment compares the four simulation policies for MAST described in Section 8.2.3: Gibbs distribution, ε-greedy, roulette wheel and UCB1 for the games Dou Di Zhu, Hearts, and Lord of the Rings: The Confrontation (LOTR:C).

To tune the parameters for the policies, between 500 and 1000 games were played for each of the domains and each of several parameter settings. For Gibbs sampling values of $\tau \in \{0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1, 1.5, 2, 4\}$ were tested, and

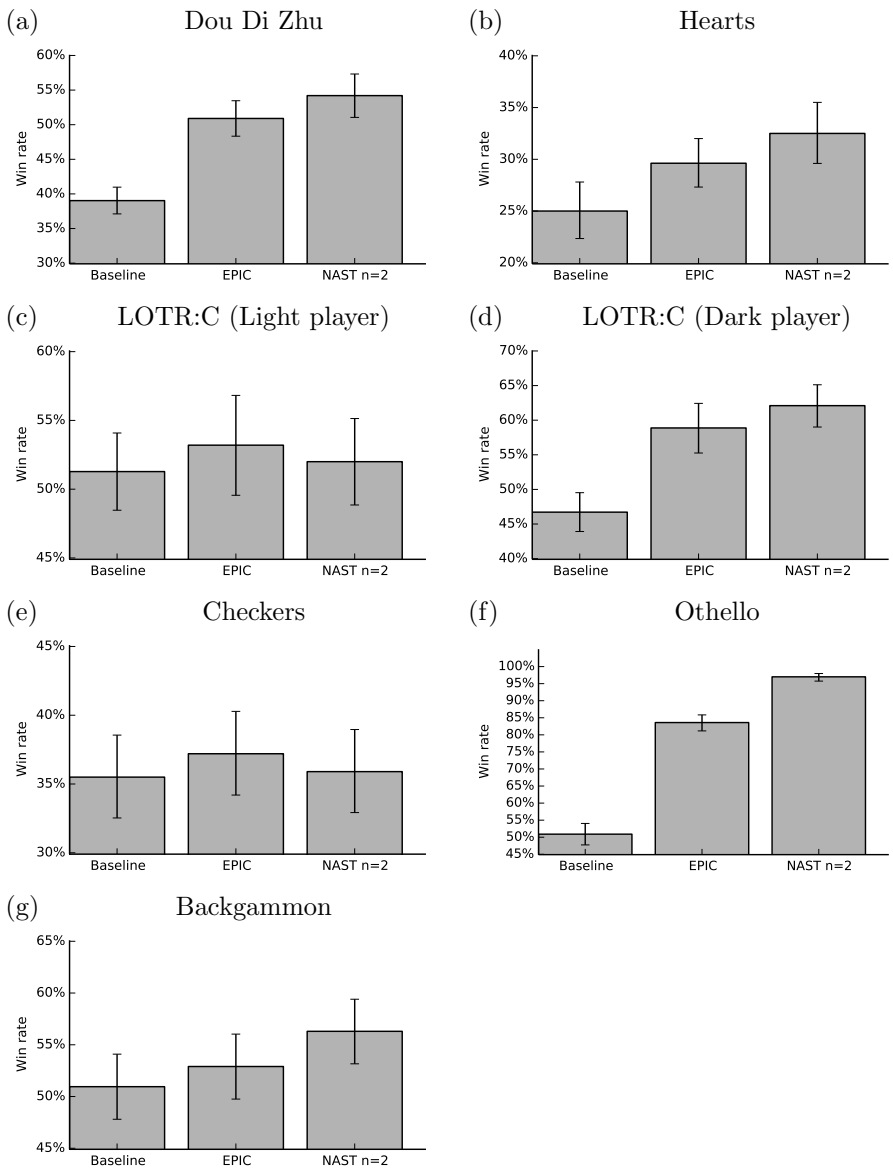

Figure 8.5: Playing strengths for a baseline player (I_{Baseline}), a player using EPIC for simulations ($I_{\text{Baseline}} \triangleright I_{\text{EPIC}}$), and a player using NAST with n-gram length 2 ($I_{\text{Baseline}} \triangleright I_{\text{NAST2}}$, with I_{NAST2} as defined in Specification 5 with $n = 2$).

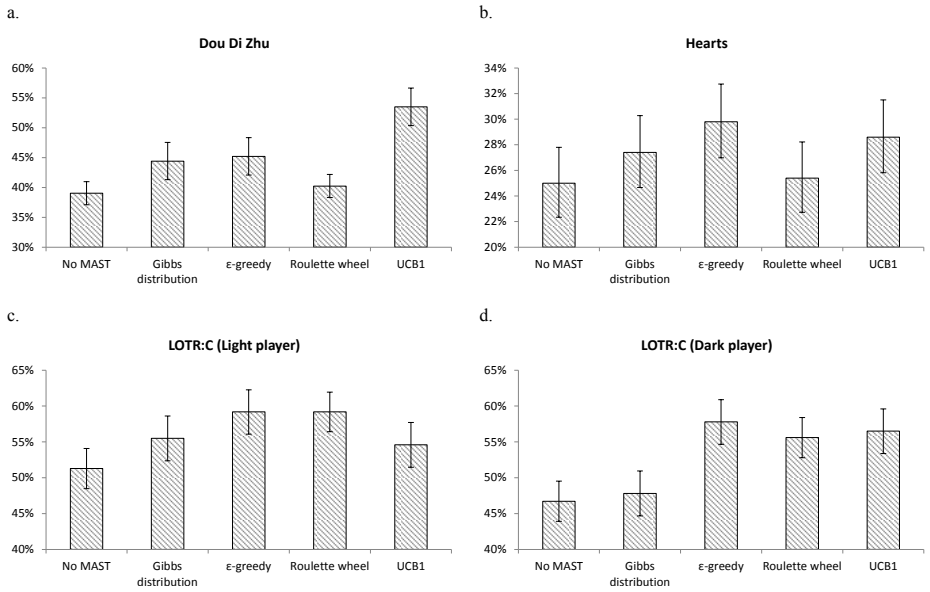

Figure 8.6: Comparison of simulation policies used by the MAST enhancement, showing percentage of games won with 95% confidence intervals. The win rate for a player not using MAST is included for reference.

$\tau = 1$ gave the strongest play across all three games. For ε-greedy, $\varepsilon \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$ was tested and $\varepsilon = 0.2$ found to be best. For UCB1 $c \in \{0.2, 0.5, 0.7, 1.0\}$ was tested and $c = 0.7$ found to give the best performance overall. Roulette wheel sampling has no parameters to tune.

For a κ-player game, one instance of the algorithm in question is played against $\kappa - 1$ instances of the baseline player (unenhanced MO-ISMCTS). Each algorithm uses 5 000 iterations per decision. For Dou Di Zhu the MAST player plays as the Landlord against two non-Landlord baseline players; for Hearts the MAST player plays against three baseline players; for LOTR:C the MAST player plays as both Light and Dark (in separate experiments) against a baseline player. Each experiment played between 500 and 2500 games.

The results of this experiment are shown in Figure 8.6. The relative strengths of the policies varies between games, but UCB1 and ε-greedy perform consistently well across all domains. UCB1 is significantly better than all other policies for Dou Di Zhu and is within experimental error of the best policies for the other games, while ε-greedy is amongst the best policies for all domains except Dou Di Zhu. Gibbs distribution sampling is significantly worse than the other policies for LOTR:C as Dark, and indeed shows no benefit over the player not using MAST in this domain. Gibbs distribution sampling does outperform the baseline in Dou Di Zhu, but does not reach the performance level of MAST with UCB1. Roulette wheel sampling performs well in LOTR:C, but fails to outperform the baseline for Dou Di Zhu and Hearts.

For Hearts, MAST has no effect on the simulation policy used for the root player's own moves: all of the cards in the player's hand must be played at some point, so every playout updates the same set of actions. Thus the average reward for an action is the average reward for all playouts, which is the same for all actions. This is not true for opponent decisions, as the opponent's cards vary between determinizations; here the average reward for an action is the average reward for all playouts on determinized hands that contain that card. This has the effect that if dealing a particular card to an opponent gives them an advantage, the opponent will be more likely to play that card earlier in subsequent playouts. The weakness of this strategy coupled with the lack of influence on the root player's strategy may explain why MAST, independent of simulation policy, is less beneficial for Hearts than for the other games. Indeed there was no statistically significant benefit to MAST for Hearts, although the fact that three of the four policies give a win rate higher than 25% suggests that significance is likely to be achieved with more trials. This argument does not apply to Dou Di Zhu and LOTR:C, where the set of all available moves is far larger than the set of moves in a particular playout.

Several authors (e.g. [142, 56, 61]) have observed that the simulation policy used by MCTS must preserve *diversity*: that is, it must strike a balance of playing plausible moves but not doing so too deterministically. A strong but deterministic simulation policy can often lead to weaker play than a less strong policy that incorporates randomness. Although it is not always phrased as such, this is an exploitation-exploration tradeoff. All of the policies considered here achieve this to some extent, but the best performing policies are those explicitly designed to mix determinism and randomness (ε-greedy) or to handle the exploitation-exploration tradeoff in multi-armed bandits (UCB1). Tak et al [63] have previously noted the strong performance of an ε-greedy policy for MAST; it has been shown that UCB1 is not significantly worse than ε-greedy, and in Dou Di Zhu is significantly better.

8.5.4 N-gram lengths

This experiment tests the strength of the NAST enhancement (Section 8.2.3) for different N-gram lengths. Having established UCB1 as a consistently strong simulation policy for MAST in the previous section, it is used exclusively in this section. The NAST player was tested under the same conditions as Section 8.5.3. The results are shown in Figure 8.7.

For LOTR:C as Dark, there is a sharp drop-off in performance between $N = 2$ and $N = 3$. For $N = 2$, NAST is maintaining average rewards for each pair consisting of an opponent move and a root player action in reply. This is a similar principle to the Last-Good-Reply Policy [60]. These results suggest that this principle is useful for LOTR:C as Dark, but longer N-grams dilute the statistics beyond the point of usefulness. This effect is not seen for LOTR:C as Light. This is in keeping with the observation in [27] that Dark must use a more reactive play style whilst Light must plan further ahead. For Hearts, $N = 2$ and $N = 3$ give significantly better performance than the baseline, whereas

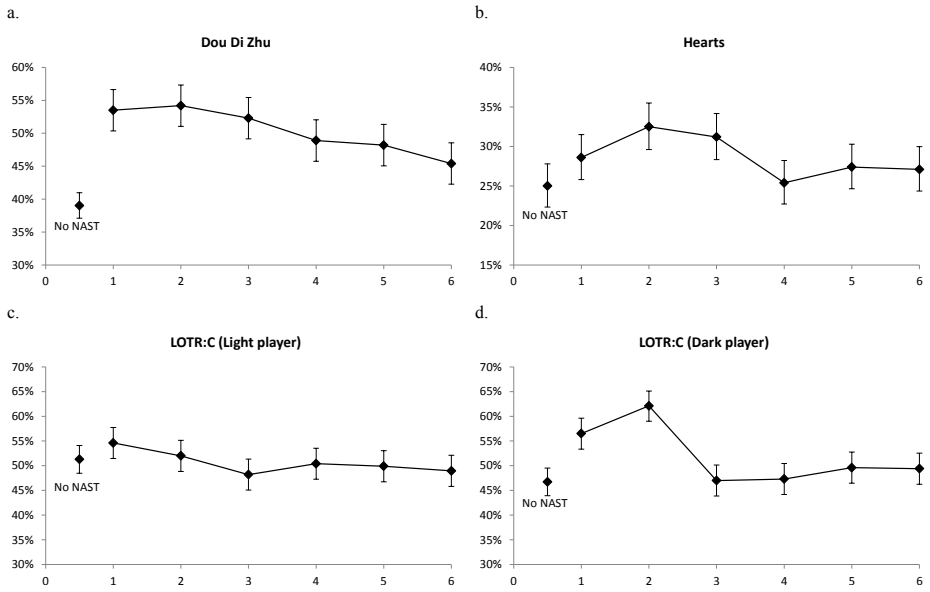

Figure 8.7: Comparison of N-gram lengths for the NAST enhancement, showing percentage of games won with 95% confidence intervals. The win rate for a player not using NAST is included for reference.

other values do not. The limited usefulness of $N = 1$ (MAST) for Hearts was discussed in Section 8.5.3. These results suggest that there is value in applying a Last Good Reply like principle, or more generally in learning the values of moves in the context of the last move or pair of moves (which in many cases means the card or pair of cards most recently played in the current trick). For Dou Di Zhu and LOTR:C as Light the performance of NAST decreases as N increases, with $N = 1$ performing best or equal best.

The main disadvantage of longer N-grams is that more iterations are required to accumulate reliable average rewards. This is illustrated in Figure 8.8: the average number of visits to each N-gram decreases exponentially as N increases. Not only does this mean that the value estimates for N-grams are less accurate for larger N, it also indicates a lower chance of encountering an N-gram that has previously been seen during simulation, thus limiting the influence of NAST. Additionally, $N = 2$ already produces a rich and expressive model for learning a simulation policy, which was as effective as the EPIC model tailored to the episodic structure of each game.

Longer N-grams may be more beneficial when the computational budget is increased. Figure 8.9 plots the playing strength of NAST for Dou Di Zhu, varying both the N-gram length and the number of iterations. The experimental conditions are as in the previous experiments, except that the unenhanced opponents use 25 000 iterations per decision. The results indicate that shorter N-grams perform better than longer ones. However Figure 8.9 also plots a line

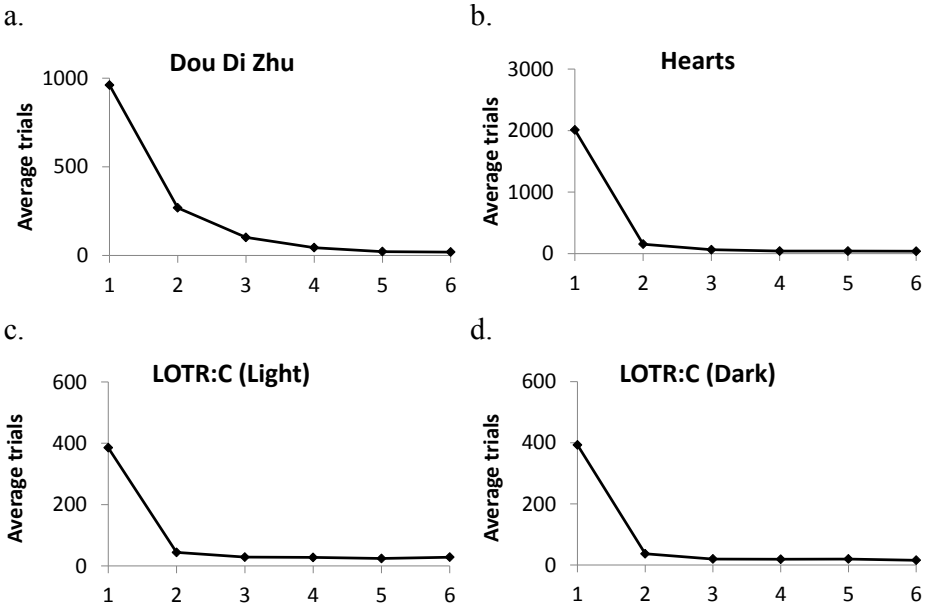

Figure 8.8: For NAST with varying N-gram lengths, these graphs plot the average number of trials accumulated for each N-gram (including only those N-grams that receive at least one trial), averaged over all decisions by one player in 500 games.

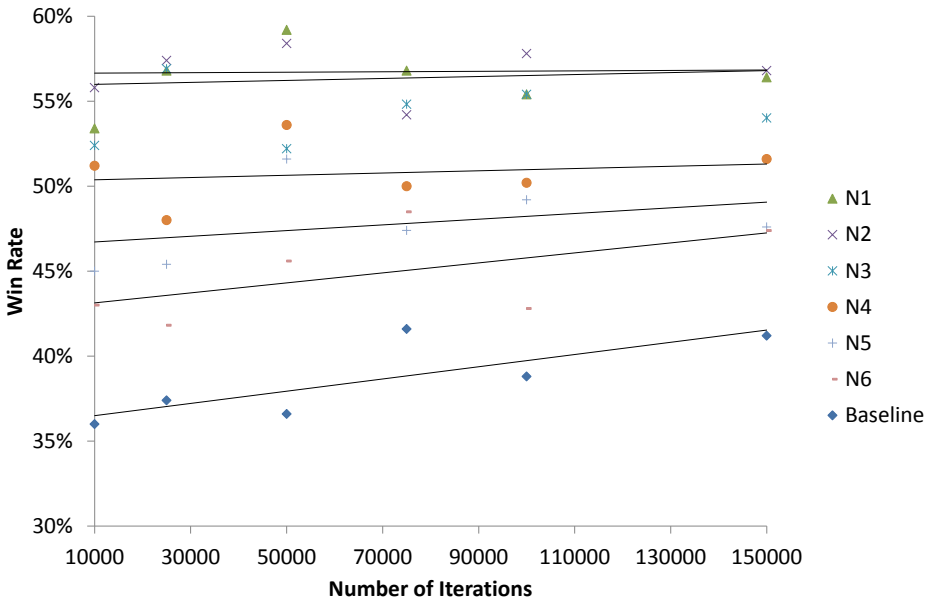

Figure 8.9: Performance of different N-gram lengths as the number of simulations varies for Dou Di Zhu as well baseline algorithm (with a random simulation policy). Also shown is the line of best fit for each N-Gram Length (using linear regression).

of best fit for each N-gram. There is a clear pattern: the rate of improvement is faster for longer N-grams than shorter ones. This is illustrated in Figure 8.10, which plots the gradient of the lines of best fit with respect to N-gram length.

These results suggests that shorter N-grams learn faster, which would be expected since they get updated more frequently, however they quickly reach a cap on performance with respect to number of iterations. Using longer N-grams resulted in strictly worse performance, but better improvement with respect to number of iterations. This suggests that the longer N-grams are continuing to learn as the number of iterations increase. It is not clear whether the longer N-grams will eventually perform better or reach the same performance limit and it would most likely require an intractably large number of MCTS iterations to reach this point (many millions of iterations would result in minutes per decision on modern hardware). If this was the case, there may be some promise in an enhancement which dynamically alters the N-gram length.

8.5.5 Computation time

The experiments in Section 8.5.1 show that certain ICARUS combinations significantly improve the playing strength of ISMCTS. However there is a computational overhead associated with the more complex policies these enhancements use. When the computational budget is specified in terms of time, the decision

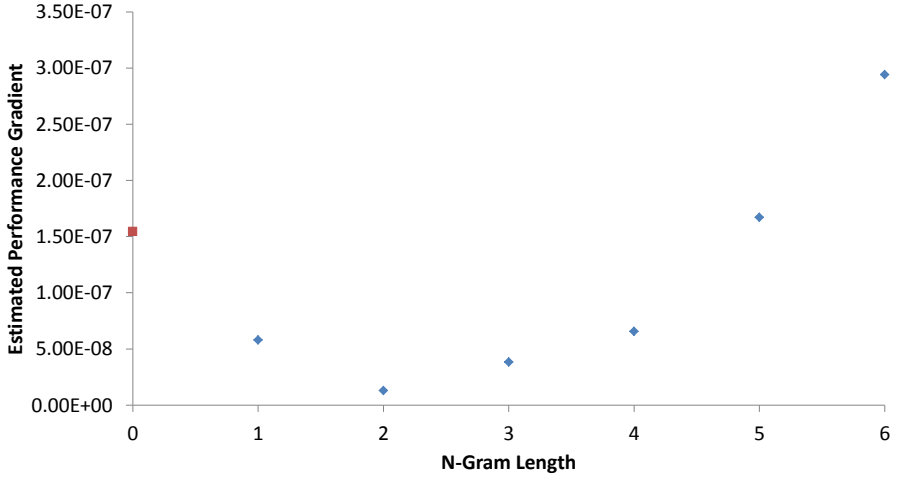

Figure 8.10: This graph shows the gradient of the lines of best fit from Figure 8.9, illustrating the relative improvment rates for each N-gram length. The gradient for the baseline algorithm is included for comparison where the N-gram length is 0.

of whether to use an enhancement is a trade-off between the value and the cost it adds to each iteration.

It was investigated whether the computational overheads are justified by the gains in playing strength for the ICARUS combinations which significantly improved the baseline algorithm in Section 8.5. This was done by allocating each agent, including the baseline opponents, a fixed budget of computational time in the range of 1 to 15 seconds per decision, rather than a fixed number of MCTS iterations.

For Dou Di Zhu it was observed that EPIC outperforms the baseline algorithm when both use 2.5 seconds or more per decision. Here 2.5 seconds corresponds to an average of 3034 iterations per decision for the baseline player and 2447 iterations per decision for EPIC. For other games and ICARUS combinations there was no benefit, but the results are obscured by the efficiency of the implementation of the algorithm. There is much more room for optimisation in the complex simulation policies of MAST, LGRF and EPIC than in the simple baseline policy of purely random moves, so the benefit of enhancements would likely be clearer in an implementation where speed was a design goal.

8.6 Summary

In this chapter the ICARUS framework was introduced, a tool for expressing information capture and reuse enhancements in MCTS. The ICARUS framework is expressive enough to enable all existing MCTS enhancements to be

defined and provides a new tool for discovery of novel types of enhancement. ICARUS provide a consistent method for expressing how an MCTS enhancement captures and reuses information, enabling easy analysis of similarities and differences between enhancements and define composition operators which are compatible with all ICARUSes. Information capture and reuse have been presented as the principles upon which all general purpose enhancements to MCTS are based, and the mechanisms for capture and reuse have been separated as a tool for understanding existing enhancements and designing new ones. The UCB1 algorithm was found to be particularly effective at balancing the exploration and exploitation of moves during MCTS simulations, leading to an elegant formulation of MCTS where UCB1 is used to select moves throughout the entire playout.

Considering which states map to which records during playout and backpropagation gives a clear indication as to which parts of the tree share information. It is likely that the effectiveness of information capture is determined by the degree of correlation of state values in these regions of the underlying game tree. In Section 8.4 it was discussed how the effectiveness of these enhancements can be explained in the context of sharing information between states.

The EPIC enhancement was developed within the ICARUS framework by considering the notion of episodes, which turns out to generalise readily to several other games. Using episodes based on the episodic structure of each game proved to be effective across each of the test domains. Many games have a natural episodic nature and EPIC may prove to be useful in exploiting this. MAST and LGR may be viewed as techniques which reuse information based on short episodes, of length 1 for MAST and length 2 for LGR. EPIC suggests that extending the notion of an episode is effective. It should be noted that NAST (with N-grams of length 2) was just as effective as EPIC, without requiring an episode structure to be defined for a particular game. This reinforces the result that episodic learning is highly effective at improving MCTS, and suggests that N-grams of length 2 are just as effective at capturing episodic information as a more sophisticated structure tailored to a particular domain. Whether EPIC out-scales NAST as more iterations are used is an open question.

The enhancements considered in this chapter are *general purpose*, in the sense that they can be applied to any game without injection of knowledge. RAVE, MAST and LGR are general purpose; strictly speaking EPIC is not, but the degree of domain knowledge required is very small. General purpose enhancements are useful tools for tackling new domains where expert knowledge is not available, and essential for domains such as general game playing where input of external knowledge is not possible. No general purpose enhancement has yet been discovered that is beneficial in all domains, and the existence of such an enhancement seems unlikely, but some are more *robust* than others; that is, they may not always be beneficial but they are seldom detrimental. Robustness is an essential criterion in choosing a general purpose enhancement. It has been demonstrated that MAST, LGR and EPIC are robust (in the test domains used in this chapter) while RAVE is not.

The ICARUS framework enables combination operators for enhancements

to be defined, with the strongest play often coming from a combination of enhancements. One possible direction for future work is to develop more robust composition operators, for example ones based on majority voting rather than weighted sums, effectively adopting an ensemble approach to enhancement combination.

One contribution of this chapter is the result that enhancements designed for perfect information games can be effective in imperfect information games, despite the increased level of uncertainty and sparsity of search (as well as the presence of subset-armed bandits in the case of ISMCTS). Current MCTS enhancements do not explicitly address information asymmetry and stochasticity in games. However new ICARUSes could be designed that consider information asymmetry in information capture, for example by sharing information between states that are distinguishable to a player but indistinguishable to their opponent. The MCTS algorithm is also easily parallelizable [143], which suggests a new class of enhancements that capture information in one search thread and reuse it in others running concurrently.

In future work it would be interesting to investigate the automation of designing and choosing enhancements for a particular game, and ICARUS provides a framework for doing this. Alternately it may be possible to measure the correlation between different areas of a game tree and use this information to select the most appropriate enhancements from a predefined library. This could be done offline before the search begins, or online to dynamically activate and deactivate enhancements as the search progresses. This kind of dynamically self-enhancing system combined with MCTS would take steps further towards a truly general purpose AI system for acting in challenging games and complex sequential decision problems.

Chapter 9

Conclusion

This thesis addressed the question: **How can *Monte Carlo Tree Search* (MCTS) deal with uncertainty and hidden information?** MCTS was extended to games with imperfect information, in order to create MCTS based algorithms which could be used to create AI players for complex board and card games whilst exhibiting challenging levels of play and without utilizing significant domain knowledge (beyond the game rules) or more than a few seconds of decision time on modern desktop hardware. The two main contribution of this work are the introduction of the *Information Set MCTS* (ISMCTS) family of algorithms, which search trees of information sets rather than trees of states and the *Information Capture and Re-Use* (ICARUS) framework which redefines MCTS algorithm in terms of exploiting knowledge learned from simulated games to improve a tree selection policy and a simulation policy. It has been shown that ISMCTS can address different sources of hidden information and uncertainty and overcome the issues of non-locality and strategy fusion (described in Chapter 3.3.1 page 3.3.1) which affect other determinization based algorithms. Additionally the ICARUS framework has been used to precisely define the implementation of both new and existing general purpose enhancements to MCTS and the effectiveness of different enhancements tested across a wide variety of games. This chapter summarizes how each of the original research hypotheses has been addressed and reflects on how effective the new approaches are and where there is scope for further work.

Hypothesis 1: Uncertain outcomes and unknown states can be represented using an MCTS tree

One of the main challenges in imperfect information games is the combinatorial explosion of possible future states caused by uncertainty. There are few popular games which have a small enough number of states to be solved exactly. Instead algorithms can make simplifying assumptions which reduce the number of states which need to be considered by a search algorithm. In a wider context of search algorithms for imperfect information games a spectrum of algorithms exist which differ according to which simplifying assumptions are

made. One one end of this spectrum are algorithms such as *Perfect Information Monte Carlo Search* (PIMC Search). PIMC search has been successful in a number of domains and can be applied to almost any domain, but by searching determinizations PIMC search suffers from the problems of strategy fusion and non-locality which leads to suboptimal play in many game (and importantly for commercial games, uninteresting play since information hiding and gathering are not considered by these algorithms). At the other end of this spectrum are approaches such as counterfactual regret minimization (CFR) which are guaranteed to converge to a Nash-equilibrium but require a policy to be stored for every information set, so may only be applied to small games or an appropriate reduction of a larger game.

ISMCTS bridges the gap between these two approaches, providing a new approach for reducing the number of nodes in the search tree by merging together histories which are not distinguishable by the root player. This solves the issue of strategy fusion in PIMC search, but learns a weaker model for opponent decisions compared to using a unique node for each unique information set. By constructing a less complex tree ISMCTS will learn faster at the cost of potentially learning suboptimal strategies. It has been demonstrated that in some games this weaker opponent model is not a hindrance to achieving a strong level in play, especially in games with the property that the value of an action is highly correlated across information sets where that action is legal. In all cases however, the ISMCTS algorithms presented in this thesis are not guaranteed to converge to a Nash-equilibrium. It is possible that ISMCTS could be modified to converge to a Nash-equilibrium by taking an approach similar to Heinrich and Silver [84], which would involve creating a unique node for each unique information set and would likely slow the learning rate of the ISMCTS algorithm. However ISMCTS has been shown to improve playing strength compared to existing approach of perfect information Monte Carlo search in the games Dou Di Zhu, Lord of the Rings: The Confrontation (LOTR:C) and the Phantom $(4, 4, 4)$ Game (see Chapters 5 and 6). Additionally ISMCTS has demonstrated to be competitive with human players in LOTR:C and Spades (see Chapters 5 and 6) and with commercially developed AI in Dou Di Zhu, Spades (see Chapters 5 and 6) and Hearts (see Chapter 8).

Hypothesis 2: The issues of strategy fusion and non-locality with determinization based approaches can solved using MCTS

A key contribution of this work is to address the main issues with determinization based methods, namely strategy fusion and non-locality. ISMCTS solves the problem of strategy fusion by design through the use of an information set tree as described in Chapter 5. Chapter 7 investigated how the issue of non-locality could be addressed in ISMCTS through the use of an inference model. It is always possible to use any external inference model with ISMCTS to bias the distribution of determinizations sampled. It was demonstrated in Chapter 7 that an inference model can be built where beliefs are calculated using ISMCTS tree statistics, and the search tree is used to filter beliefs. Another issue with determinization approaches such as PIMC search is that bluffing is impossible.

This was addressed in Chapter 7 by using self-determinizations with ISMCTS. Together the combination of inference and self-determinization with ISMCTS solves the issue of non-locality whilst preserving the generality of ISMCTS.

Hypothesis 3: Inference and bluffing can be modeled using MCTS

In Chapter 7 ISMCTS was modified to exploit an opponent model, where beliefs were drives from statistics in the ISMCTS tree which are subsequently used to bias the sampling of determinizations. This contributes the first attempt at building a belief model for MCTS directly from MCTS statistics, in contrast to approaches based on particles [82] or offline fixed models [49]. This technique allows inference to be performed automatically using ISMCTS and it was demonstrated to be effective at guessing the identity of the Spies in the game The Resistance. However in games such as The Resistance, players attempt to conceal information through bluffing which determinization based approaches cannot do (since there is no information to hide). To induce bluffing behaviour, several methods for injecting self-determinizations were tested. This was very effective in the game The Resistance, where self-determinization techniques were able to significantly mitigate against opponents using inference. These techniques for inference and bluffing hinder the learning rate of the ISMCTS algorithm. As a consequence, performing inference in the more complex games Scotland Yard and Saboteur is difficult without access to an infeasible amount of computational resources. ISMCTS could be significantly improved in these domains by including heuristics which bucket together different beliefs in order to improve the learning rate. It is also possible that sharing information between simulations could be used to improve the learning of an opponent model in a similar manner to the ICARUS framework.

Hypothesis 4: Knowledge captured from simulations be exploited to create better MCTS policies

In Chapter 8 the ICARUS framework was introduced, a tool for expressing information capture and reuse enhancements in MCTS, where information is captured from Monte Carlo simulations, and can be reused to improve the MCTS tree selection and simulation policies. This contributes the first cohesive framework for describing all of the common techniques used for modifying MCTS implementations. The ICARUS framework is expressive enough to enable all existing MCTS enhancements to be defined and provides a new tool for discovery of novel types of enhancement. Several new and existing enhancements were tested in a variety of combinations across many domains. It was observed that some enhancements are *robust* (as in they may not always be beneficial but they are seldom detrimental). Choosing robust enhancements is important when creating general purpose MCTS algorithms.

9.1 Remarks on the Effectiveness of ISMCTS

ISMCTS achieves one of the main goals of this project, to make competitive AI for complex board and card cards using a small amount of decision time on modern hardware. Recently ISMCTS has been integrated into a commercial game Spades by AI Factory [10] for the android mobile operating system. The existing knowledge-based AI for Spades was already a market leader, and generally recognised as strong. The ISMCTS-based player performs better by a statistically significant margin, playing at or above a level for the previous AI which consumed over 8 times as much computation time. ISMCTS play is particularly strong in the challenging case of nil bid play. Additionally the ISMCTS implementation now constitutes a ready-made AI module that can simply be dropped into future projects and produce a strong level of play with almost no modification. Indeed, AI Factory are already using the same code for new games currently in development. The ISMCTS framework also provides extensive hooks for game-specific knowledge injection to improve both objective playing strength and subjective playing style.

ISMCTS does have a few drawbacks, most notably that ISMCTS has no convergence guarantees and in some instances may converge to suboptimal policies which has been noted by other authors [66, 87]. In Chapter 7 a version of ISMCTS was tested which was designed to allow bluffing behavior to emerge from search, by learning policies for all players information sets. This has the effect of reducing the exploitability of ISMCTS. However a large number of iterations of the algorithm are required for this behaviour to emerge removing one of the most useful properties of ISMCTS, that a strong level of performance can be achieved with only a few seconds of decision time. It is possible to consider a version of ISMCTS which accurately models the game being played, by placing a bandit algorithm at every information set in the game. This algorithm would not converge to a Nash-equilibrium due to the limitations of the UCB algorithm and the lack of self-determinization (see-below). Bandit based algorithms do not typically reach an equilibrium and may exhibit behaviour such as oscillating between different strategies. However Heinrich and Silver have shown that the UCB1 algorithm can be modified to converge towards a Nash-equilibrium when applied to a tree which contains one unique node per unique information set [84]. If approximating a Nash-equilibrium is necessary for reasonable play in a game then ISMCTS could be modified in a similar way. However ISMCTS is well suited to quickly finding robust pure strategies if they exist using just the standard UCB1 algorithm for selection.

Another drawback of the ISMCTS algorithm is the lack of self-determinization, that is the assumption that other players can observe the information set of the searching player. This is akin to assuming other players can see your hand in a card game. In Chapter 7 it is shown that very large number of iterations are required for strong play to emerge when self-determinizations are included in the search. It was also observed in Chapter 6 that the lack of self determinization leads ISMCTS to be overly pessimistic when enough iterations are used. This is one area where ISMCTS has the potential to improve with more

powerful hardware, which would enable a more accurate model of the game to be searched which included self determinizations and different nodes for opponent information sets (to avoid the subset armed bandits at opponent nodes). A version of ISMCTS could be developed which uses coarse groupings of information sets (as with single observer ISMCTS) when an information set has been visited a small number of times but transition into fine grained grouping of information sets (a node of every information set belonging to every player) as the number of iterations increase. This would produce an algorithm which continues to scale as larger number of iterations are used, but is as effective as possible when smaller numbers of iterations are used. In addition a one node per information set tree (like that used by Heinrich and Silver [84]) could be updated in parallel to ISMCTS, then this tree used for selection once the nodes have received a sufficiently large number of updates. This algorithm have the best of both worlds: The fast learning and scaling properties of the information set grouping techniques of ISMCTS, but with convergence to a Nash-equilibrium given a sufficiently large amount of computation.

One key observation from the experiments presented in this thesis, is that a competitive level of play can be achieved in many popular games, whilst effectively ignoring some aspects of uncertainty. For example consider the results presented in Chapter 5 for the game Mini Dou Di Zhu. One experiment showed that in many situations, a player knowing information that could not be reasonably deduced often does not give the player an advantage. In these situations there are many states within an information set, but strategies exist which will win the game regardless of the underlying state of the game. This suggests that a strong level of play can be achieved in these games whilst making simplifying assumption about the hidden information aspect of the game. In particular such games are particularly suited to ISMCTS, which is well suited to exploited a line of play which leads to a high probability of winning. It is likely that the presence of effective pure strategies in many popular imperfect information games is a consequence of game design and a human preference for games where these types of strategy exist.

There are several ISMCTS algorithms in this thesis and they differ by how information sets are mapped onto trees. Effectively each of these algorithms is using a different structure to learn information from simulated games and guide tree selection. The ICARUS framework was initially developed to generalise the many different enhancements to MCTS by classifying them according to how information is captured from simulated games and used to improve either the tree selection policy or the simulation policy of MCTS. Therefore ISMCTS can be thought of as a class of enhancements to MCTS for imperfect information games. MCTS as a concept is hard to define, since nearly every application of MCTS is different some way. However the concept of information capture from simulated games and subsequent re-use in a tree selection policy and simulation policy defined in the ICARUS framework is shared between all MCTS algorithms. Therefore it can be concluded that combination of those three components, simulated games, a tree selection policy and a simulation policy are the true essence of MCTS algorithm. The ICARUS framework therefore is a use-

ful tool for describing MCTS algorithms, since the mechanism for information capture and subsequent re-use in the algorithm is explicitly drawn out from the definition of an MCTS algorithm.

In conclusion, ISMCTS is a new family of algorithms for search in games with hidden information and uncertainty which has been demonstrated to be more effective than existing approaches in a variety of domains. ISMCTS is a generic algorithm like MCTS, and can be tailored to different games in order to balance the accuracy of the model searched by ISMCTS with the rate at which ISMCTS converges. ISMCTS is particularly suited to large complex games of imperfect information where robust pure strategies exist and there is some evidence that this is often the case (at least for the games studied in this thesis). Finally the ICARUS framework defines MCTS algorithms in terms of information capture from simulated games and subsequent reuse in tree and simulation policies. This facilitates the comparison of different MCTS algorithms and enhancements and was used to determine the most effective combination of enhancements across a wide variety of domains.

9.2 Future Work

The following is a summary of interesting directions for extending this work.

- ISMCTS converges faster when searching a simpler model of the game, but if more computational resources are available a more precise model of the game can be searched where an independent multi-armed bandit is used for every information set of every player which would produce better strategies. Additionally ISMCTS could be improved by adjusting the precision of the model as more iterations are performed, resulting in an algorithm which produces the strongest possible level of play for any number of iterations.

- Currently ISMCTS does not have any convergence guarantees. One direction for future work is to understand what sort of strategy ISMCTS does converge towards by characterising the equilibrium formed by bandit algorithms in an adversarial setting. For example, a bandit algorithm is capable of exploiting a dominating strategy (that receives a greater average reward than some other strategy regardless of opponent actions). This can then be used to describe properties of games that will indicate that ISMCTS will be successful (for example the existence of good pure strategies).

- There is already some evidence that ISMCTS produces AI players which humans enjoy playing against and that knowledge can be included with ISMCTS to make AI players which appear plausible to human players. Further work could investigate in more detail what sort of AI behaviour is fun to play against and how to reproduce this behaviour within ISMCTS.

It was observed when integrating ISMCTS into a commercial Spades engine [10] that ISMCTS fails to play plausibly when losing (or sometimes winning) a game, since all actions appear to have equal utility. Similarly ISMCTS offers potentially infinitely tunable difficulty levels through adjusting the number of iterations, but must appear to make plausible mistakes for the best player experience.

- There are many games which are too large and complex to produce plausible behaviour using ISMCTS and no domain knowledge. However it is often the case that knowledge exists or can be generated (through machine learning for example). There is a large body of work describing methods of integrating knowledge with MCTS [1]. However further work can be done to investigate what sort of knowledge mostly effectively improves ISMCTS, in particular knowledge pertaining to aspects of information asymmetry and stochasticity which are not present in the perfect information games previously studied with MCTS. ICARUS provides a framework for investigating the automation of designing and choosing enhancements for a particular game. It would be interesting to explore extensions of ICARUS which allow knowledge injection. Additionally, it may be possible to automatically discover new enhancements using evolutionary techniques; the ICARUS framework could give a compact yet expressive representation for genetic programming or other evolutionary algorithms.

Bibliography

[1] C. Browne, E. J. 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," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6145622

[2] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Euro. Conf. Mach. Learn.*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Germany: Springer, 2006, pp. 282–293. [Online]. Available: http://www.springerlink.com/index/d232253353517276.pdf

[3] G. M. J.-B. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Monte-Carlo Strategies for Computer Go," in *Proc. BeNeLux Conf. Artif. Intell.*, Namur, Belgium, 2006, pp. 83–91. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.8924\&rep=rep1\&type=pdf

[4] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Proc. 5th Int. Conf. Comput. and Games, LNCS 4630*, Turin, Italy, 2007, pp. 72–83. [Online]. Available: http://portal.acm.org/citation.cfm?id=1777826.1777833

[5] C.-S. Lee, M.-H. Wang, G. M. J.-B. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Trans. Comp. Intell. AI Games*, vol. 1, no. 1, pp. 73–89, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4804732

[6] C.-S. Lee, M. Müller, and O. Teytaud, "Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 225–228, Dec. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5371786

[7] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," *Artif. Intell.*, vol. 175,

no. 11, pp. 1856–1875, Jul. 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S000437021100052X

[8] Y. Björnsson and H. Finnsson, "CadiaPlayer: A Simulation-Based General Game Player," *IEEE Trans. Comp. Intell. AI Games*, vol. 1, no. 1, pp. 4–15, 2009. [Online]. Available: http://www.ru.is/kennarar/yngvi/pdf/BjornssonF09.pdf

[9] J. Méhat and T. Cazenave, "A Parallel General Game Player," *Künstliche Intelligenz*, vol. 25, no. 1, pp. 43–47, 2011. [Online]. Available: http://www.springerlink.com/index/A83GT77455J377K8.pdf

[10] D. Whitehouse, P. I. Cowling, E. J. Powley, and J. Rollason, "Integrating Monte Carlo Tree Search with Knowledge-Based Methods to Create Engaging Play in a Commercial Mobile Game," in *Proc. Artif. Intell. Interact. Digital Entert. Conf.*, Boston, Massachusetts, 2013, pp. 100–106.

[11] J. Von Neumann, O. Morgenstern, A. Rubinstein, and H. Kuhn, *Theory of games and economic behavior*. Princeton Univ Pr, 2007. [Online]. Available: http://books.google.co.uk/books?hl=en\&lr=\&id=QeN8-NA9K_sC\&oi=fnd\&pg=PR27\&dq=theory+of+games+and+economic+behavior\&ots=BSF5xMCe9M\&sig=Q9PnNsd1Dd4OcCueNnY3pCKBzyY

[12] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[13] C. Shannon, "XXII. Programming a computer for playing chess," *Philosophical Magazine (Series 7)*, vol. 41, no. 314, pp. 256–275, 1950. [Online]. Available: http://scholar.google.co.uk/scholar?rlz=1C1GPCK_enGB406GB406\&q=Programming+a+Computer+for+Playing+Chess\&um=1\&ie=UTF-8\&sa=N\&hl=en\&tab=ws\#0

[14] J. Madziuk, *Knowledge-Free and Learning-Based Methods in Intelligent Game Playing*, ser. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 276. [Online]. Available: http://link.springer.com/10.1007/978-3-642-11678-0

[15] T. Kozelek, "Methods of MCTS and the game Arimaa," M.S. thesis, Charles Univ., Prague, 2009.

[16] M. L. Ginsberg, "GIB: Imperfect Information in a Computationally Challenging Game," *J. Artif. Intell. Res.*, vol. 14, pp. 303–358, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.9118\&rep=rep1\&type=pdf

[17] B. Sheppard, "World-championship-caliber Scrabble," *Artif. Intell.*, vol. 134, pp. 241–275, 2002.

[18] B. Abramson, "Expected-Outcome: A General Model of Static Evaluation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, pp. 182 – 193, 1990.

[19] F.-H. Hsu, *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion.* Princeton, New Jersey: Princeton Univ. Press, 2004.

[20] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers Is Solved," *Science*, vol. 317, pp. 1518–1522, 2007.

[21] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo Tree Search in Hex," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 251–258, 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5551182

[22] N. Ikehata and T. Ito, "Monte-Carlo Tree Search in Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 39–46.

[23] T. Pepels and M. H. M. Winands, "Enhancements for Monte-Carlo Tree Search in Ms Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 265–272.

[24] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Monte Carlo Tree Search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 234–241.

[25] D. Perez, E. J. Powley, D. Whitehouse, P. Rohlfshagen, S. Samothrakis, P. I. Cowling, and S. M. Lucas, "Solving the Physical Travelling Salesman Problem: Tree Search and Macro-Actions," *IEEE Trans. Comp. Intell. AI Games (submitted)*, 2013.

[26] D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and Information Set Monte Carlo Tree Search for the Card Game Dou Di Zhu," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 87–94.

[27] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information Set Monte Carlo Tree Search," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 2, pp. 120–143, 2012.

[28] I. Frank and D. Basin, "Search in games with incomplete information: a case study using Bridge card play," *Artif. Intell.*, vol. 100, no. 1-2, pp. 87–123, 1998.

[29] M. L. Ginsberg, "GIB: Steps Toward an Expert-Level Bridge-Playing Program," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 16. Stockholm, Sweden: Citeseer, 1999, pp. 584–593. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.3210\&rep=rep1\&type=pdf

[30] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search," in *Proc. Assoc. Adv. Artif. Intell.*, Atlanta, Georgia, 2010, pp. 134–140. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/viewPDFInterstitial/1876/1942

[31] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Bandits all the way down: UCB1 as a simulation policy in Monte Carlo Tree Search," in *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, Ontario, Canada, 2013, pp. 81–88.

[32] ——, "Monte Carlo Tree Search with Macro-Actions and Heuristic Route Planning for the Multiobjective Physical Travelling Salesman Problem," in *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, Ontario, Canada, 2013, pp. 73–80.

[33] R. B. Myerson, *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

[34] J. Schaeffer, *One Jump Ahead: Computer Perfection at Checkers*. Berlin, Germany: Springer, 1997.

[35] D. Koller, N. Megiddo, and B. von Stengel, "Fast Algorithms for Finding Randomized Strategies in Game Trees," in *Proc. ACM Symp. Theory Comput.*, vol. 54, Montreal, Canada, 1994, pp. 750–759. [Online]. Available: http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:No+Title\#0

[36] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret Minimization in Games with Incomplete Information," in *Proc. Adv. Neur. Inform. Process. Sys.*, Vancouver, Canada, 2008, pp. 1729–1736. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.421\&rep=rep1\&type=pdf

[37] M. Ponsen, M. Lanctot, and S. de Jong, "MCRNR: Fast Computing of Restricted Nash Responses by Means of Sampling," in *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, Atlanta, Georgia, 2010. [Online]. Available: http://www.aaai.org/ocs/index.php/WS/AAAIW10/paper/viewFile/1985/2463

[38] C. J. Clopper and E. S. Pearson, "The use of confidence or fiducial limits illustrated in the case of the binomial," *Biometrika*, vol. 26, no. 4, pp. 404–413, 1934.

[39] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002. [Online]. Available: http://www.springerlink.com/index/L7V1647363415H1T.pdf

[40] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron," in *Proc. IEEE Conf. Comput. Intell. Games*, Granada, Spain, 2012, pp. 242–249.

[41] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proc. Annu. Symp. Found. Comput. Sci.*, Milwaukee, Wisconsin, 1995, pp. 322–331. [Online]. Available: http://www.computer.org/portal/web/csdl/doi/10.1109/SFCS.1995.492488

[42] O. Teytaud and S. Flory, "Upper Confidence Trees with Short Term Partial Information," in *Proc. Applicat. Evol. Comput. 1, LNCS 6624*, C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, J. J. M. Guervós, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. N. Yannakakis, Eds., Torino, Italy, 2011, pp. 153–162.

[43] B. Brügmann, "Monte Carlo Go," Max-Planke-Inst. Phys., Munich, Tech. Rep., 1993. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.449\&rep=rep1\&type=pdf

[44] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with Patterns in Monte-Carlo Go," Inst. Nat. Rech. Inform. Auto. (INRIA), Paris, Tech. Rep., 2006.

[45] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *Proc. Adv. Neur. Inform. Process. Syst.*, Vancouver, Canada, 2006. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.7504\&rep=rep1\&type=pdf

[46] B. Arneson, R. B. Hayward, and P. Henderson, "MoHex Wins Hex Tournament," *Int. Comp. Games Assoc. J.*, vol. 32, no. 2, pp. 114–116, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.2612\&rep=rep1\&type=pdf

[47] R. B. Segal, "On the Scalability of Parallel UCT," in *Proc. Comput. and Games, LNCS 6515*, Kanazawa, Japan, 2010, pp. 36–47.

[48] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower Bounding Klondike Solitaire with Monte-Carlo Planning," in *Proc. 19th Int. Conf. Automat. Plan. Sched.*, Thessaloniki, Greece, 2009, pp. 26–33. [Online]. Available: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/download/724/1090

[49] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, "Improving State Evaluation, Inference, and Search in Trick-Based Card Games," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, Pasadena, California, 2009, pp. 1407–1413. [Online]. Available: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI-09/paper/download/474/908

[50] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Determinization in Monte-Carlo Tree Search for the card game Dou Di Zhu," in *Proc. Artif. Intell. Simul. Behav.*, York, United Kingdom, 2011, pp. 17–24.

[51] C. D. Ward and P. I. Cowling, "Monte Carlo Search Applied to Card Selection in Magic: The Gathering," in *Proc. IEEE Symp. Comput. Intell. Games*, Milan, Italy, 2009, pp. 9–16. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5286501

[52] P. I. Cowling, C. D. Ward, and E. J. Powley, "Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 4, pp. 241–257, 2012.

[53] R. Ramanujan, A. Sabharwal, and B. Selman, "On the Behavior of UCT in Synthetic Search Spaces," in *Proc. 21st Int. Conf. Automat. Plan. Sched.*, Freiburg, Germany, 2011. [Online]. Available: http://www.informatik.uni-freiburg.de/~icaps11/proceedings/mcts/ramanujan-et-al.pdf

[54] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *Proc. 24th Annu. Int. Conf. Mach. Learn.* Corvalis, Oregon: ACM, 2007, pp. 273–280. [Online]. Available: http://portal.acm.org/citation.cfm?id=1273496.1273531

[55] H. Finnsson and Y. Björnsson, "Simulation Control in General Game Playing Agents," in *Proc. Int. Joint Conf. Artif. Intell. Workshop Gen. Game Playing*, Pasadena, California, 2009, pp. 21–26. [Online]. Available: http://www2.ru.is/kennarar/yngvi/pdf/FinnssonB09a.pdf

[56] G. M. J.-B. Chaslot, C. Fiter, J.-B. Hoock, A. Rimmel, and O. Teytaud, "Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search," in *Proc. Adv. Comput. Games, LNCS 6048*, Pamplona, Spain, 2010, pp. 1–13. [Online]. Available: http://www.springerlink.com/index/5906866772N5K427.pdf

[57] H. Nguyen, K. Ikeda, and B. Le, "Extracting important patterns for building state-action evaluation function in Othello," in *IEEE Conf. Technologies and Applications of Artificial Intelligence (TAAI)*, 2012, pp. 278–283.

[58] P. D. Drake and S. Uurtamo, "Heuristics in Monte Carlo Go," in *Proc. Int. Conf. Artif. Intell.*, Las Vegas, Nevada, 2007, pp. 171–175. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.7460\&rep=rep1\&type=pdf

[59] H. Finnsson and Y. Björnsson, "Simulation-Based Approach to General Game Playing," in *Proc. Assoc. Adv. Artif. Intell.*, Chicago, Illinois, 2008, pp. 259–264. [Online]. Available: http://www.aaai.org/Papers/AAAI/2008/AAAI08-041.pdf

[60] P. D. Drake, "The Last-Good-Reply Policy for Monte-Carlo Go," *ICGA Journal*, vol. 32, no. 4, pp. 221–227, 2009.

[61] H. Baier and P. D. Drake, "The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 303–309, 2010.

[62] J. A. Stankiewicz, M. H. M. Winands, and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search Enhancements for Havannah," in *Proc. 13th Int. Conf. Adv. Comput. Games, LNCS 7168*, Tilburg, The Netherlands, 2012, pp. 60–71.

[63] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 2, pp. 73–83, 2012.

[64] BoardGameGeek, "Lord of the Rings: The Confrontation." [Online]. Available: http://boardgamegeek.com/boardgame/3201/lord-of-the-rings-the-confrontation

[65] M. Shafiei, N. R. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in Simultaneous Games," in *Proc. Int. Joint Conf. Artif. Intell. Workshop Gen. Game Playing*, Pasadena, California, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.161.6138http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.6138\&rep=rep1\&type=pdf

[66] M. Lanctot, V. Lis, and M. H. M. Winands, "Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel," in *Proceedings of IJCAI 2013 Worksop on Computer Games*, 2013.

[67] C. Luckhart and K. B. Irani, "An Algorithmic Solution of N-Person Games," *AAAI*, vol. 86, pp. 158–162, 1986.

[68] N. R. Sturtevant and R. E. Korf, "On Pruning Techniques for Multi-Player Games," in *Proc. Assoc. Adv. Artif. Intell./Innov. Applic. Artif. Intell.*, Austin, Texas, 2000, pp. 201–207.

[69] N. R. Sturtevant, "An Analysis of UCT in Multi-Player Games," in *Proc. Comput. and Games, LNCS 5131*, Beijing, China, 2008, pp. 37–49. [Online]. Available: http://www.springerlink.com/index/t15q7487968415n5.pdf

[70] S. Samothrakis, D. Robles, and S. M. Lucas, "Fast Approximate Max-n Monte-Carlo Tree Search for Ms Pac-Man," *IEEE Trans. Comp. Intell. AI Games*, vol. 3, no. 2, pp. 142–154, 2011.

[71] T. Cazenave, "Multi-player Go," in *Proc. Comput. and Games, LNCS 5131*, Beijing, China, 2008, pp. 50–59. [Online]. Available: http://www.springerlink.com/index/b1670p817770707j.pdf

[72] J. A. M. Nijssen and M. H. M. Winands, "Monte Carlo Tree Search for the Hide-and-Seek Game Scotland Yard," *IEEE Trans. Comp. Intell. AI Games*, vol. 4, no. 4, pp. 282–294, Dec. 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6266709

[73] ——, "Enhancements for Multi-Player Monte-Carlo Tree Search," in *Proc. Comput. and Games, LNCS 6515*, Kanazawa, Japan, 2011, pp. 238–249.

[74] E. K. P. Chong, R. L. Givan, and H. S. Chang, "A Framework for Simulation-based Network Control via Hindsight Optimization," in *Proc. IEEE Conf. Decis. Cont.*, Sydney, Australia, 2000, pp. 1433–1438.

[75] J. Schäfer, "The UCT Algorithm Applied to Games with Imperfect Information," Diploma thesis, Otto-Von-Guericke Univ. Magdeburg, Germany, 2008.

[76] J. Borsboom, J.-T. Saito, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, "A Comparison of Monte-Carlo Methods for Phantom Go," in *Proc. BeNeLux Conf. Artif. Intell.*, Utrecht, Netherlands, 2007, pp. 57–64. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.7715\&rep=rep1\&type=pdf

[77] P. Ciancarini and G. P. Favini, "Monte Carlo tree search in Kriegspiel," *Artif. Intell.*, vol. 174, no. 11, pp. 670–684, Jul. 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370210000536

[78] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey: Prentice Hall, 2009. [Online]. Available: http://books.google.com/books?hl=en\&lr=\&id=8jZBksh-bUMC\&pgis=1

[79] B. W. Ballard, "The *-Minimax Search Procedure for Trees Containing Chance Nodes," *Artif. Intell.*, vol. 21, no. 3, pp. 327–350, Sep. 1983. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(83)80015-0

[80] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron, "Game-Tree Search with Adaptation in Stochastic Imperfect-Information Games," in *Proc. Int. Conf. Comput. and Games, LNCS 3846*, Ramat-Gan, Israel, 2004, pp. 21–34. [Online]. Available: http://www.springerlink.com/index/y80710288171754j.pdf

[81] S. J. J. Smith and D. S. Nau, "Strategic Planning for Imperfect-Information Games," Institute for Systems Research, Tech. Rep., 1993.

[82] D. Silver and J. Veness, "Monte-Carlo Planning in Large POMDPs," in *Proc. Neur. Inform. Process. Sys.*, Vancouver, Canada, 2010, pp. 1–9.

[83] T. Furtak and M. Buro, "Recursive Monte Carlo Search for Imperfect Information Games," in *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, Ontario, Canada, 2013, pp. 225–232.

[84] J. Heinrich and D. Silver, "Self-Play Monte-Carlo Tree Search in Computer Poker," in *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, Québec, Canada, 2014, pp. 19–25.

[85] D. Koller and A. Pfeffer, "Representations and solutions for game-theoretic problems," *Artif. Intell.*, vol. 94, no. 1-2, pp. 167–215, 1997. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370297000234

[86] N. A. Risk and D. Szafron, "Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents," in *Proc. 9th Int. Conf. Auton. Agents Multi. Sys.*, Toronto, Canada, 2010, pp. 159–166.

[87] M. Lanctot, V. Lisý, and M. Bowling, "Search in Imperfect Information Games using Online Monte Carlo Counterfactual Regret Minimization," in *Proceedings of the AAAI Workshop on Computer Poker and Imperfect Information*, no. Sandholm 2010, 2014. [Online]. Available: http://mlanctot.info/files/papers/aaai14-w4-iioos.pdf

[88] J. Rubin and I. Watson, "Computer poker: A review," *Artif. Intell.*, vol. 175, no. 5-6, pp. 958–987, Apr. 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370211000191

[89] M. Ponsen, G. Gerritsen, and G. M. J.-B. Chaslot, "Integrating Opponent Models with Monte-Carlo Tree Search in Poker," in *Proc. Conf. Assoc. Adv. Artif. Intell.: Inter. Decis. Theory Game Theory Workshop*, Atlanta, Georgia, 2010, pp. 37–42.

[90] M. Richards and E. Amir, "Opponent Modeling in Scrabble," in *Proc. 20th Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 1482–1487. [Online]. Available: http://www.aaai.org/Papers/IJCAI/2007/IJCAI07-239.pdf

[91] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 4–10.

[92] Pagat, "Dou Dizhu." [Online]. Available: http://www.pagat.com/climbing/doudizhu.html

[93] B. E. Childs, J. H. Brodeur, and L. Kocsis, "Transpositions and Move Groups in Monte Carlo Tree Search," in *Proc. IEEE Symp. Comput. Intell. Games*, Perth, Australia, 2008, pp. 389–395. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5035667

[94] Pagat, "Hearts." [Online]. Available: http://www.pagat.com/reverse/hearts.html

[95] N. R. Sturtevant and A. M. White, "Feature Construction for Reinforcement Learning in Hearts," in *Proc. Comput. and Games*, 2006, pp. 122–134.

[96] Pagat, "Spades." [Online]. Available: http://www.pagat.com/boston/spades.html

[97] BoardGameGeek, "Stratego." [Online]. Available: http://boardgamegeek.com/boardgame/1917/stratego

[98] J. W. H. M. Uiterwijk and H. J. van den Herik, "The advantage of the initiative," *Inform. Sci.*, vol. 122, no. 1, pp. 43–58, Jan. 2000. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S002002559900095X

[99] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck, "Games solved: Now and in the future," *Artif. Intell.*, vol. 134, no. 1-2, pp. 277–311, Jan. 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370201001527

[100] L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens, "Go-Moku Solved By New Search Techniques," *IEEE Comput. Intell. Mag.*, vol. 12, no. 1, pp. 7–23, Feb. 1996. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-8640.1996.tb00250.x

[101] D. Auger, "Multiple Tree for Partially Observable Monte-Carlo Tree Search," in *Proc. Evol. Games.*, Torino, Italy, 2011, pp. 53–62. [Online]. Available: http://www.springerlink.com/index/25G28J028841161U.pdf

[102] F. Teytaud and O. Teytaud, "Lemmas on Partial Observation, with Application to Phantom Games," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 243–249.

[103] P. Ciancarini and G. P. Favini, "Monte Carlo Tree Search Techniques in the Game of Kriegspiel," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, Pasadena, California, 2009, pp. 474–479. [Online]. Available: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI-09/paper/download/396/693

[104] H. Finnsson and Y. Björnsson, "Learning Simulation Control in General Game-Playing Agents," in *Proc. 24th AAAI Conf. Artif. Intell.*, Atlanta, Georgia, 2010, pp. 954–959. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/download/1892/2124

[105] M. Buro, "From Simple Features to Sophisticated Evaluation Functions," in *Proc. Comput. and Games, LNCS 1558*, Tsukuba, Japan, 1998, pp. 126–145. [Online]. Available: http://www.springerlink.com/index/3VRH9JNFCGAM1MQX.pdf

[106] D. Robles, P. Rohlfshagen, and S. M. Lucas, "Learning Non-Random Moves for Playing Othello: Improving Monte Carlo Tree Search," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 305–312.

[107] P. Hingston and M. Masek, "Experiments with Monte Carlo Othello," in *Proc. IEEE Congr. Evol. Comput.*, Singapore, 2007, pp. 4059–4064. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4425000

[108] J. A. M. Nijssen, "Playing Othello Using Monte Carlo," B.S. Thesis, Maastricht Univ., Netherlands, 2007.

[109] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, 1995. [Online]. Available: http://www.informatik.uni-osnabrueck.de/barbara/lectures/selforganization/papers/tdgammon.ps.gz

[110] F. van Lishout, G. M. J.-B. Chaslot, and J. W. H. M. Uiterwijk, "Monte-Carlo Tree Search in Backgammon," in *Proc. Comput. Games Workshop*, Amsterdam, Netherlands, 2007, pp. 175–184. [Online]. Available: http://www.montefiore.ulg.ac.be/~vanlishout/publications/vanlishout_backgammon.pdf

[111] BoardGameGeek, "The Resistance." [Online]. Available: http://boardgamegeek.com/boardgame/41114/the-resistance

[112] ——, "Scotland Yard." [Online]. Available: http://boardgamegeek.com/boardgame/438/scotland-yard

[113] J. A. M. Nijssen and M. H. M. Winands, "Monte-Carlo Tree Search for the Game of Scotland Yard," in *Proc. IEEE Conf. Comput. Intell. Games*, Seoul, South Korea, 2011, pp. 158–165. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6032002

[114] BoardGameGeek, "Saboteur." [Online]. Available: http://boardgamegeek.com/boardgame/9220/saboteur

[115] S. Yoon, A. Fern, R. L. Givan, and S. Kambhampati, "Probabilistic Planning via Determinization in Hindsight," in *Proc. Assoc. Adv. Artif. Intell.*, Chicago, Illinois, 2008, pp. 1010–1017. [Online]. Available: http://www.aaai.org/Papers/AAAI/2008/AAAI08-160.pdf

[116] M. Ponsen, J. Ramon, T. Croonenborghs, K. Driessens, and K. Tuyls, "Bayes-Relational Learning of Opponent Models from Incomplete Information in No-Limit Poker," in *Proc. Assoc. Adv. Artif. Intell.*, Chicago, Illinois, 2008, pp. 1485–1486.

[117] C. A. Luckhardt and K. B. Irani, "An Algorithmic Solution of N-Person Games," in *Proc. Assoc. Adv. Artif. Intell.*, Philadelphia, Pennsylvania, 1986, pp. 158–162. [Online]. Available: http://scholar.google.co.uk/scholar?hl=en\&q=luckhardt+algorithmic\&um=1\&ie=UTF-8\&sa=N\&tab=ws\#0

[118] H. Kuhn, "A Simplified Two-Person Poker," in *Contributions to the Theory of Games*, H. Kuhn and A. Tucker, Eds. Princeton University Press, 1950, pp. 97–103.

[119] M. Ponsen, S. de Jong, and M. Lanctot, "Computing Approximate Nash Equilibria and Robust Best-Responses Using Sampling," *J. Artif. Intell. Res.*, vol. 42, pp. 575–605, 2011.

[120] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, "Monte Carlo Sampling for Regret Minimization in Extensive Games," in *Proc. Adv. Neur. Inform. Process. Sys.*, Vancouver, Canada, 2009, pp. 1078–1086. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.4143\&rep=rep1\&type=pdf

[121] M. Lanctot, "Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games," Ph.D. thesis, University of Alberta, 2013.

[122] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artif. Intell.*, vol. 6, no. 4, pp. 293–326, 1975.

[123] S. G. Akl and M. M. Newborn, "The Principal Continuation and the Killer Heuristic," in *Proc. ACM Annu. Conf.*, 1977, pp. 466–473.

[124] J. Schaeffer, "The History Heuristic and Alpha-Beta Search Enhancements in Practice," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 11, pp. 1203–1212, 1989. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=42858

[125] R. E. Korf, "Depth-first iterative-deepening: an optimal admissible tree search," *Artif. Intell.*, vol. 27, no. 1, pp. 97–109, 1985.

[126] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5288526

[127] S. Thrun, "Is Learning The n-th Thing Any Easier Than Learning The First?" in *Proc. Neur. Inform. Process. Sys.*, 1996, pp. 640–646.

[128] R. Caruana, "Multitask Learning," *Mach. Learn.*, vol. 28, pp. 41–75, 1997.

[129] R. Vilalta and Y. Drissi, "A Perspective View and Survey of Meta-Learning," *Artif. Intell. Rev.*, vol. 18, pp. 77–95, 2002.

[130] D. Tom and M. Müller, "Computational Experiments with the RAVE Heuristic," in *Proc. Comput. and Games, LNCS 6515*, Kanazawa, Japan, 2010, pp. 69–80. [Online]. Available: http://www.springerlink.com/index/501143511Q207634.pdf

[131] R. E. Korf, "Multi-player alpha-beta pruning," *Artif. Intell.*, vol. 48, no. 1, pp. 99–111, Feb. 1991. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/000437029190082U

[132] D. P. Helmbold and A. Parker-Wood, "All-Moves-As-First Heuristics in Monte-Carlo Go," in *Proc. Int. Conf. Artif. Intell.*, Las Vegas, Nevada, 2009, pp. 605–610. [Online]. Available: http://users.soe.ucsc.edu/~dph/mypubs/AMAFpaperWithRef.pdf

[133] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Math. Nat. Comput.*, vol. 4, no. 3, pp. 343–357, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.9239\&rep=rep1\&type=pdf

[134] A. Rimmel and F. Teytaud, "Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search," in *Proc. Applicat. Evol. Comput. 1, LNCS 6624*, Torino. Italy, 2010, pp. 201–210. [Online]. Available: http://www.springerlink.com/index/J768144866443031.pdf

[135] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte-Carlo Tree Search Solver," in *Proc. Comput. and Games, LNCS 5131*, Beijing, China, 2008, pp. 25–36. [Online]. Available: http://www.springerlink.com/index/a520n00375m70771.pdf

[136] ——, "Monte Carlo Tree Search in Lines of Action," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 239–250, 2010.

[137] Pagat, "Card Games: Climbing Games." [Online]. Available: http://www.pagat.com/climbing/

[138] I. Chernev, *Combinations: The Heart of Chess*. Dover Publications, 1967.

[139] Sensei's Library, "Joseki," 2012. [Online]. Available: http://senseis.xmp.net/?Joseki

[140] ——, "Tesuji," 2012. [Online]. Available: http://senseis.xmp.net/?Tesuji

[141] A. Rimmel, O. Teytaud, C.-S. Lee, S.-J. Yen, M.-H. Wang, and S.-R. Tsai, "Current Frontiers in Computer Go," *IEEE Trans. Comp. Intell. AI Games*, vol. 2, no. 4, pp. 229–238, 2010.

[142] D. Silver and G. Tesauro, "Monte-Carlo Simulation Balancing," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Montreal, Canada, 2009, pp. 945–952. [Online]. Available: http://portal.acm.org/citation.cfm?id=1553495

[143] G. M. J.-B. Chaslot, M. H. M. Winands, and H. J. van den Herik, "Parallel Monte-Carlo Tree Search," in *Proc. Comput. and Games, LNCS 5131*, Beijing, China, 2008, pp. 60–71. [Online]. Available: http://www.springerlink.com/index/u39846x66k535218.pdf