Selective Search in Games of Different Complexity

Selective Search in Games of Different Complexity

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de Rector Magnificus,
Prof. mr. G.P.M.F. Mols,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op woensdag 25 mei 2011 om 16.00 uur

door

Maarten Peter Dirk Schadd

Promotor: Prof. dr. G. Weiss
Copromotor: Dr. M.H.M. Winands
Dr. ir. J.W.H.M. Uiterwijk

Leden van de beoordelingscommissie:
Prof. dr. ir. R.L.M. Peeters (voorzitter)
Dr. Y. Björnsson (Reykjavik University)
Prof. dr. H.J.M. Peters
Prof. dr. ir. J.A. La Poutré (Universiteit Utrecht)
Prof. dr. ir. J.C. Scholtes

Netherlands Organisation for Scientific Research

This research has been funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project TACTICS, grant number 612.000.525.

Dissertation Series No. 2011-16
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by Optima Grafische Communicatie, Rotterdam

ISBN 978-94-6169-060-9

Preface

As far as I can think back, I have enjoyed playing all kind of games. A fascinating property of games is that with simple rules, very complex behavior can be created. This behavior includes facets such as long-term planning, tactical decisions and setting up traps for the opponent. Once people find a job, their interest in games makes place for a more serious life. This has not happened to me yet. I still enjoy playing games on a regular basis and have no intention in the near future to change this. Over the years, my game collection has grown quite a bit, and some of those games which I discovered, are made part of this thesis. I even managed to obtain the title of Dutch Board Game Champion in 2009. Therefore it seems adequate that I have chosen a profession where games play a central role. This thesis is a result of a 27 years fascination for games.

First of all, I would like to thank my daily supervisor Mark Winands for his stimulating efforts over the last years. He did not only channel my thoughts into scientific publications, but also helped me to avoid dangerous pitfalls in research. I also have to thank him for his endless patience and his deep knowledge on search methods. Next, many thanks go to my supervisor Jos Uiterwijk who gave me the opportunity to get acquainted to research in games for my Master's thesis. Without him I would not have continued my games research in the shape of a Ph.D. thesis. I also want to thank Gerhard Weiss, who agreed to be my promotor, and Lena Kurzen, who was my partner at the NWO TACTICS project. This project was headed by Prof. Dr. Johan van Benthem.

Moreover, I would like to thank all those people with whom I have collaborated over the past years. I enjoyed writing articles with Jaap van den Herik, Guillaume Chaslot, Maurice Bergsma, Huib Aldewereld, and Jan Stankiewicz. I also want to thank the following colleagues and friends for their inspiring discussions over the past years: Sander Bakkes, Jahn-Takeshi Saito, Nyree Lemmens, Steven de Jong, Michael Kaisers, Philippe Uyttendaele, Marc Ponsen, István Szita, Pim Nijssen, Gian Piero Favini, David Lupien St-Pierre, Laurens van der Maaten, Loes Braun, Sander Spek, Femke de Jonge, Hendrik Baier, Andra Waagmeester, Stijn Vanderlooy, Mandy Tak, Sander Arts and Jesper Mohnen. A special thanks goes to Peter Geurtz who was making sure that my experiments kept running on the cluster. I furthermore want to thank the secretaries Marijke Verheij and Joke Hellemons who helped me to find my way in the administrative labyrinth.

In order to be able to focus on work, you have to distract yourself from work regularly. Here I want to thank those people who helped me to recharge my batteries

from time to time. First I would like to thank the members of the Slimbo Spelgroep Limburg, especially Alex Peters, Juanita Vernooy, Pieter Spronck, Wim van Gruisen and Marcel Falise, for many nights full of new and exciting board games. Second, I would like to thank the many members of the student cycling association Dutch Mountains for many scenic hours in the Limburgian scenery. Third, I also thank Andreas Hofmann, Achim Hofmann, Raphaela Hofmann, Dirk Zander, Andreas Schebesta, Sarah Schebesta, Markus Stahl, Nico Simon and others for providing even more opportunities for distraction.

I want to thank my parents, Peter and Anneke, for allowing me to realize my own potential. Further thanks go to Frederik, Brigitte and Kurt for their support in my adventures. Na koniec chciałbym podziękować Klaudynie za jej miłość i troskę.

Maarten Schadd, 2010

Acknowledgments

The research has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems. This research has been funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project TACTICS, grant number 612.000.525.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

This thesis investigates how selective-search methods can improve the performance of a game program for a given domain. Selective-search methods aim to explore only the profitable parts of the state space, but take the risk to overlook the best move. We propose several selective-search methods and test them in a large number of game domains. The domains consist of deterministic one-, two- and multi-player games with perfect information, and two-player non-deterministic or imperfect-information games.

In this chapter we provide a brief introduction on games research (Section 1.1) and discuss selective search for games in Section 1.2. Next, we formulate the problem statement which guides our research, together with four research questions (Section 1.3). Finally, we provide an overview of this thesis in Section 1.4.

1.1 Games

Since thousands of years, games are a phenomenon across human cultures, where people display intelligence, interaction and competition (Huizinga, 1955; Bell, 1980). But games are also an important theoretic paradigm in logic, AI, computer science, linguistics, economics, biology, and increasingly also in social sciences and social psychology. Games can be classified according to different dimensions. Five classifications are (1) the number of players (one-, two-, multi-player), (2) whether chance is involved (deterministic or non-deterministic), (3) how much information a player has (perfect or imperfect information), (4) which time system is used (turn-based or real-time), and (5) the decision space (discrete or continuous). Dimensions (4) and (5) are relevant for video games which are beyond the scope of the thesis. For the remainder of this section, games are turn-based and discrete, if not mentioned otherwise.

What makes games of particular interest is their hybrid character. On the one hand, the rules of the game are well-defined, the states are easy to represent and the possible actions are known, but on the other hand, games allow complex behavior and reasoning. One example of a simple game which still forms a challenge is the L-Game (De Bono, 1968), where players are moving an L-shaped piece on a 4×4

board, trying to block the opponent from moving. Therefore, games are an ideal AI testbed for better understanding intelligent human behavior including decision-making, intentions, strategies, and interaction. Study of these topics has a rich tradition in logic and computer science. Here, cognition meets computation: games challenge humans by their difficulty, and the study of machines by humans highlights basic issues of complexity.

The outline of this section is as follows. In Subsection 1.1.1 we describe some research performed on games from the mathematical and logical viewpoint. Afterwards, in Subsection 1.1.2, the computational approach to games is discussed.

1.1.1 Game Theory

Game theory can be defined as the study of mathematical models of conflict and cooperation between intelligent rational decision-makers (cf. Myerson, 1997; Myerson, 1999; Peters, 2008). Mathematical game theory provides general techniques for analyzing game-like situations in which two or more players make decisions that will influence one's another's welfare. It has a wide range of applications, but is most prominently seen in economics. Pioneering work has been done by Zermelo (1913), Borel (1921) and Von Neumann and Morgenstern (1944). In this field, several research efforts were successful to the point of giving rise to research areas of their own, of which we want to introduce two. (1) Game-theoretic semantics for logics (Lorenz and Lorenzen, 1978) aim at understanding a game on a conceptual level. Meaning is given to the components of the game, such as the number of players, the goal of a player and the interaction between the players. Depending on the game, different types of logic may be used, such as modal logic, epistemic logic and dynamic-epistemic logic (Van Benthem, 2001). (2) Combinatorial game theory (CGT) is a mathematical theory that studies finite two-player deterministic games with perfect information, such as chess, checkers, Go, Domineering, Dots-and Boxes, Nim, and many others (Nowakowski, 2009). Berlekamp, Conway, and Guy (1982) use a stricter definition of a combinatorial game, which adds that a player unable to move loses the game (*normal play*) and that no draws exist because of move repetition (*ending condition*) (Guy, 1996). The aim of CGT is to find the algebraic structure of the game such that the optimal play for both players can be determined. Additionally, this field of research is concerned with the complexity of games. A game falls into a certain complexity class, depending on how fast the problem grows when the input size is varied. Two of these classes are called PSPACE-complete and EXPTIME-complete. Many games are proven to be PSPACE-complete (Papadimitriou, 1994; Van Emde Boas, 2002), such as Go-Moku (Reisch, 1980), Hex (Reisch, 1981), Othello (Iwata and Kasai, 1994), Scotland Yard (Sevenster, 2006), Connect6 (Hsieh and Tsai, 2007) and Amazons (Hearn, 2009). Games that are proven to be EXPTIME-complete include Chinese Checkers (Kasai, Adachi, and Iwata, 1979), chess (Fraenkel and Lichtenstein, 1981), Go (Robson, 1983), checkers (Robson, 1984) and Shogi (Adachi, Kamekawa, and Iwata, 1987). For an overview of a large number of games and their complexity classes we refer to Demaine and Hearn (2001).

1.1.2 Games and AI

Since the dawn of computer technology, AI researchers have let computers play games as a testing ground for computational intelligence. The AI research in games obtained an impulse in 1944 when Von Neumann republished his article about the minimax algorithm (Von Neumann, 1928) together with Morgenstern in the book "Theory of Games and Economic Behavior" (Von Neumann and Morgenstern, 1944). These ideas were continued by Shannon (1950), Turing (1953), and Newell, Shaw, and Simon (1958) who wrote the first articles about how computers could play chess as intelligently as possible.

A popular research area in the field of games and AI concerns two-player deterministic perfect-information games. A classic example of these games is chess. A great deal of effort has been devoted in the past to construct a world-champion-level computer chess player. The most prominent success so far in this area was the result when Deep Blue defeated the world chess champion Garry Kasparov in 1997 (Hsu, 2002). The next step would be to solve the game (i.e., knowing the game-theoretic value), but Chess will not be solvable in a foreseeable future according to Levy and Newborn (1991). Not long ago, another breakthrough was achieved by Schaeffer *et al.* (2007), who solved the game of checkers. A forward search enhanced with endgame databases proved that if both players play optimally the game ends in a draw. For the game of Go, programs are not on grandmaster level yet. However, in 2009 a 9-dan Go professional was beaten by the program MoGo with "only" a 7-stone advantage.[1]

In *non-deterministic* games[2] an element of chance is involved. A well studied non-deterministic two-player game is Backgammon, in which computers are stronger than humans since the 1990's (Tesauro, 1994). One of the most famous programs, TD-Gammon by Gerald Tesauro, employs neural networks (Fukushima, 1975) and temporal-difference learning (Sutton, 1988) to achieve a high level of play. *Imperfect-information* games hide information from the players. Examples of imperfect-information games are the chess-variant Kriegspiel and the Go-variant Phantom Go, in which the pieces of the opponent are hidden. In Kriegspiel, the strongest programs use metapositions during the search, but they cannot beat the best humans (Ciancarini and Favini, 2007). In Phantom Go, the strongest programs are based on Monte-Carlo methods, and are able to play on an experienced level (Cazenave, 2006; Borsboom *et al.*, 2007). Scrabble[3] is a game with both non-determinism and imperfect information. The program Maven uses four different search engines for each phase of the game (i.e., opening, middle game, pre-endgame and endgame) with a selective move generator, a well-tuned evaluation function and Monte-Carlo simulations to beat all human Scrabble experts (Sheppard, 2002a; Sheppard, 2002b).

One-player games are essentially optimization problems, of which two famous examples are the 15-puzzle (Korf, 1985) and the Rubik's Cube (Korf, 1997). These

[1] The game was played at the Taiwan Open 2009 and can be downloaded from: http://go.nutn.edu.tw/2009/English/result_eng.htm

[2] These games may also be called stochastic games, or games with chance.

[3] Scrabble® is a registered trademark. All intellectual property rights in and to the game are owned in the U.S.A and Canada by Hasbro Inc., and throughout the rest of the world by J.W. Spear & Sons Limited of Maidenhead, Berkshire, England, a subsidiary of Mattel Inc. Mattel and Spear are not affiliated with Hasbro.

games are typically tackled with the A* algorithm (Hart, Nielson, and Raphael, 1968). One-player games may also be non-deterministic, such as Tetris (Demaine, Hohenberger, and Liben-Nowell, 2003). In Tetris, an evaluation function tuned by the Cross-Entropy Method (Rubinstein, 2003) is able to remove more than 35 million lines on average (Thiery and Scherrer, 2009). Per definition, deterministic one-player games with imperfect information do not exist. In order to have imperfect information either an element of chance should be present (e.g., shuffling of cards) and/or a second player is required (e.g., choosing a hidden card of the second player), which would contradict the notion of a one-player deterministic game. A well-known one-player non-deterministic game with imperfect information is Klondike Solitaire.[4] Here, most research focuses on the solvability of Klondike Solitaire with an initial setup of cards. Bjarnason, Tadepalli, and Fern (2007) demonstrate empirically that no less than 82% and no more than 91.44% of Klondike Solitaire games have winning solutions.

As the name already indicates, in multi-player games more than two players participate in the game. Chinese Checkers is a well-known deterministic perfect-information game in this category, which has received quite some attention from researchers (Sturtevant, 2003a). Chinese Checkers has been used as test domain for tree-search methods with coalition forming (Sturtevant and Korf, 2000) as well as for Monte-Carlo methods (Sturtevant, 2008a). Examples of multi-player games with chance are Monopoly[5] and Ludo (Carter, 2007). Visiting each square on the Monopoly board may be modeled as a Markov chain (Ash and Bishop, 1972) and evolutionary algorithms can be used to learn strategies (Frayn, 2005). Siguo (Xia *et al.*, 2005; Lu and Xia, 2008) and Scotland Yard (Sevenster, 2006) are multi-player games with imperfect information. For Siguo, Bayesian networks may be used in which beliefs are propagated (Xia, Zhu, and Lu, 2007). In Scotland Yard, Doberkat, Hasselbring, and Pahl (1996) use a multi-agent system to establish cooperation between the agents. Many multi-player card games have both non-determinism and imperfect information, such as Poker (Billings *et al.*, 1998a; Billings *et al.*, 1999) and Bridge (Ginsberg, 1999). Here, opponent modeling plays a central role (Jansen, 1992; Carmel and Markovitch, 1993; Iida *et al.*, 1994; Billings *et al.*, 1998b; Donkers, 2003).

Table 1.1 gives examples of games for each classification. We remark that a large number of multi-player games may also be played by only two players. Depending on the class of a game, different methods are necessary to create a strong AI. Taking the AI perspective into account, it is possible to use different classes. An example of an alternative classification is the theme of the game, which has a large influence for constructing an evaluation function. The theme of a game can be material (chess), territory (Go), connection (Hex), or racing (Chinese Checkers).

Finally, we mention that the field of modern video games has become increasingly prominent in games research over the last decades (cf. Bakkes, 2010). These games typically are of synchronous and continuous nature which means that a fast decision has to be made in real-time for an infinite[6] action space. This makes the creation

[4] It is just called Solitaire on the Microsoft Windows operating system.
[5] Monopoly® is a registered trademark of Hasbro, Inc. All rights reserved.
[6] or several magnitudes larger than in abstract games

Table 1.1: Games of different classifications.

	perfect information	
Players	deterministic	non-deterministic
1	15-puzzle	Tetris
2	chess, Go	Backgammon
>2	Chinese Checkers	Ludo, Monopoly
	imperfect information	
Players	deterministic	non-deterministic
1	-	Klondike Solitaire
2	Kriegspiel, Phantom Go	Two-Player Scrabble
>2	Siguo, Scotland Yard	Poker, Bridge

of video game AI a challenging task. In modern games, the intelligent behavior of game characters is often established on the basis of hacks and cheats (Millington, 2006). When observing this kind of behavior, human players often assume intelligence where none exists (Crawford, 1984; Lidén, 2004). They appreciate if the game AI maintains the illusion of being intelligent (Scott, 2002). The research on video game AI has moved from rule-based systems (Nareyek, 2000; Tozour, 2002), to goal-oriented behavior (Millington, 2006) and adaptive game AI (Spronck, 2005; Bakkes, 2010).

1.2 Selective Search

AI researchers have proposed a large number of computational methods for game playing. Applying search methods to turn-based games has been successful for many years. The most prominent search method is the *minimax* algorithm (Von Neumann, 1928), enhanced with $\alpha\beta$ *pruning* (Knuth and Moore, 1975). This algorithm searches through the state space of the game until a fixed depth is reached and evaluates the desirability of game states on the search horizon. $\alpha\beta$ pruning detects states which do not have an influence on the value of the root and prunes these states safely, increasing the search speed. $\alpha\beta$ is a form of backward pruning because a refutation is found after a move is searched. Minimax often forms the basis for other algorithms as well. Examples are *expectimax* (Michie, 1966), which introduces chance nodes to model chance events in non-deterministic games, and the *paranoid* algorithm (Sturtevant and Korf, 2000) for multi-player games.

When a computer program has to decide which move to play, it is generally limited by a time constraint. Thus, the program should utilize the given time as well as possible. In a traditional $\alpha\beta$ program with iterative deepening, the search depth is iteratively increased until time runs out. This is an iterative *depth-first search*. If domain knowledge is available, there often exists an indication whether a move is strong or weak (e.g., capturing or losing a queen in chess). Using this knowledge, *variable-depth search* can be performed (Marsland and Björnsson, 2001) that uses $\alpha\beta$ search as foundation. Variable-depth search consists of two parts: (1)

forward pruning and (2) search extensions. If a move is regarded as unlikely based on some criteria, it can be pruned. This is called forward pruning, because the decision to skip a move is made in advance. In contrast, moves which seem promising can be granted additional search depth, which is called a search extension.

While variable-depth search uses more time for strong moves and considerable less time for weak moves, the nominal search depth still has to be finished before a deeper search starts. Other approaches try to spend their time on the most interesting part of the search tree at any point of time. If nodes on the search horizon are ordered so that the one with the best evaluation is expanded first, the resulting strategy is called *best-first search* (Russell and Norvig, 2003). The search tree created by best-first search is typically narrow and deep. An example is *Proof-Number (PN) search* (Allis, Van der Meulen, and Van den Herik, 1994). PN search is a (mate-)solver which tries to prove or disprove a certain goal for the root node. This algorithm repeatedly selects the most-proving node for expansion. Therefore, PN search is always spending its time on the most interesting part of the tree. A disadvantage of PN search is that it can only be used for solving. The played move is weak if it is chosen before the position has been solved.

*SSS** (based on state-space search) is another form of best-first search of which it is proven that it expands fewer nodes than $\alpha\beta$ (Stockman, 1979). SSS* is an effective technique if no good move ordering is available because multiple solution trees are investigated at the same time. A disadvantage is, however, that a list of nodes has to be sorted and kept in memory. The additional memory requirements and sorting effort may be an overhead if a good move ordering is available.

A different type of best-first search that has become increasingly popular for letting computers play games, is *Monte-Carlo Tree Search* (MCTS) (Kocsis and Szepesvári, 2006; Chaslot *et al.*, 2006b; Coulom, 2007a). Instead of using a heuristic evaluation function to guide the search, MCTS utilizes Monte-Carlo simulations. MCTS balances exploration and exploitation to spend most effort on the best move, while making sure that alternatives are not overlooked. Recently, this technique has revolutionized computer Go (Coulom, 2007a; Gelly and Silver, 2007; Chaslot *et al.*, 2008d). MoGo and Crazy Stone, the two pioneering MCTS programs, managed to defeat professional players with less than a 10-stone handicap for the first time in history (Chaslot *et al.*, 2008a). MCTS has also been applied successfully in quite some other two-player games such as Amazons (Lorentz, 2008; Kloetzer, Iida, and Bouzy, 2009), Hex (Cazenave and Saffidine, 2009; Arneson, Hayward, and Henderson, 2010) and Lines of Action (LOA) (Winands and Björnsson, 2010).

Thus, current research is not focused on the traditional iterative depth-first approach anymore, but on approaches which spend the majority of thinking time on the parts of the tree which are the most profitable. If the search method selectively investigates the state space of a game with a risk of overlooking the best move, we call it a *selective search* (cf. Marsland, 1986; Buro, 1995; Björnsson, 2002).

1.3 Problem Statement and Research Questions

In the previous section we discussed the relevance of selective search to improve computer game play. That is precisely the topic of this thesis. The following problem statement guides our research.

> **Problem statement**: *How can we improve selective-search methods in such a way that programs increase their performance in domains of different complexity?*

Rather than testing selective-search methods on one class of games, we chose different classes of games, which all have to be addressed differently. Each class of games represents a level of complexity. Between every level there exists a *complexity jump*. With a complexity jump the complexity of the game increases significantly because the mechanism of the game is changed (Van Emde Boas, 2003) (e.g., a player, chance, or imperfect information is added). We have chosen five different levels of games, resulting in four complexity jumps. (1) One-player games, or puzzles, involve no opponent and are a testbed for planning algorithms. (2) Two-player games are the classic testbed for search methods. We use them for investigating mate-solvers. For testing search with chance nodes, (3) non-deterministic and (4) imperfect-information games may be used. (5) Multi-player games are a testbed for dealing with coalition forming.

We formulate four research questions to guide our research. Each one deals with search for a different class of games and a different selective-search method. The four research questions address (1) MCTS, (2) PN search, (3) expectimax, and (4) multi-player search.

> **Research question 1**: *How can we adapt Monte-Carlo Tree Search for a one-player game?*

The traditional approach to deterministic one-player games with perfect information is applying A* or IDA*. These methods have been quite successful in coping with this class of games. The disadvantage of these methods is that they require an admissible heuristic evaluation function. The construction of such a function can be difficult. Since Monte-Carlo Tree Search (MCTS) does not require an admissible heuristic, it may be an interesting alternative. Therefore, we investigate which modifications are required to apply MCTS successfully to a deterministic one-player game with perfect information. We call the resulting variant *Single-Player Monte-Carlo Tree Search* (SP-MCTS) and test it in the puzzle SameGame.

> **Research question 2**: *How can we solve a two-player game by using Proof-Number search in combination with endgame databases?*

Ideally, a search method is able to prove that a move is the optimal one for a given game. The game is solved if this is achieved. In the last years quite some deterministic two-player games with perfect information have been solved (e.g., Connect-Four (Allis, 1988), Nine Men's Morris (Gasser, 1996) and checkers (Schaeffer *et al.*, 2007)). A selective-search method specially designed as mate-solver is Proof-Number (PN)

search (Allis *et al.*, 1994). PN search is efficient in searching game trees with a non-uniform branching factor. Endgame databases (Ströhlein, 1970; Van den Herik and Herschberg, 1985) proved to be vital to the strength of computer programs in quite a number of games (Schaeffer, 1997; Heinz, 1999). Schaeffer *et al.* (2007) used a combination of proof-tree manager, the $\alpha\beta$ based program CHINOOK, the PN-search variant Df-pn (Nagai, 2002) and endgame databases to solve the game of checkers. We investigate to which degree endgame databases are able to enhance the PN-search variant PN^2 for the deterministic two-player perfect-information game Fanorona. The complexity of Fanorona is similar to the complexity of checkers.

> **Research question 3**: *How can we perform forward pruning at chance nodes in the expectimax framework?*

Another form of searching selectively in two-player deterministic games with perfect information is variable-depth search (Marsland and Björnsson, 2001). Branches can be pruned if they seem unpromising (forward pruning), or extended if the branches are promising (search extensions). There exist several successful forward-pruning techniques for the $\alpha\beta$ algorithm (Beal, 1989; Buro, 1995; Buro, 2000; Björnsson and Marsland, 2001). For two-player games with non-determinism or imperfect information expectimax may be used (Michie, 1966). Expectimax adds chance nodes to the search tree. There are, however, no forward-pruning techniques for chance nodes. We propose a new forward-pruning technique based on *ProbCut* (Buro, 1995), called *ChanceProbCut*, and test it in two-player games with either non-determinism or imperfect information.

> **Research question 4**: *How can we improve search for multi-player games?*

In deterministic two-player games with perfect information, the majority of research focused on the $\alpha\beta$ algorithm (Knuth and Moore, 1975). For deterministic multi-player games with perfect information, the choice of algorithm is not as straightforward. The two main algorithms are called *max*^{n} (Luckhardt and Irani, 1986) and *paranoid* (Sturtevant and Korf, 2000), both approaching the problem from a different angle. Max^n assumes that every player tries to maximize the own score, while paranoid assumes that all opponents form a coalition against the root player. However, these assumptions have drawbacks. Due to the lack of safe pruning in max^n only a limited lookahead is possible (Sturtevant, 2003b). Furthermore, the underlying assumption of max^n may be unrealistic, resulting in max^n to be too optimistic (Zuckerman, Felner, and Kraus, 2009). When searching deep with the paranoid algorithm, the other players may dominate the root player (Saito and Winands, 2010), resulting in paranoid to be too pessimistic. We propose a new search method, called *Best-Reply Search* (BRS) that avoids these drawbacks. This method does not allow every opponent to make a move, but only the one with the strongest counter move.

1.4 Thesis Overview

The thesis structure is as follows. Chapter 1 provides an introduction to the research presented in this thesis.

Chapter 2 is a general introduction to search methods for games. It explains the minimax algorithm and the well-known $\alpha\beta$ search. Standard techniques for enhancing the $\alpha\beta$ search are discussed as well. We furthermore explain Monte-Carlo Tree Search (MCTS) and its enhancements.

In Chapter 3 we answer the first research question. The chapter explains the rules of the test domain, the puzzle SameGame, and thereafter discusses the modifications made to MCTS for deterministic one-player games with perfect information. The new algorithm is called Single-Player Monte-Carlo Tree Search (SP-MCTS). We provide experiments on tuning the parameters and show that randomized restarts can improve the score. Experiments reveal that SP-MCTS is able to achieve a high score on the standardized test set.

Chapter 4 starts with explaining the rules of Fanorona, a deterministic perfect-information two-player game. Next, an analysis of this game is presented. Subsequently, the construction of omniscient endgame databases is described. The mate-solver Proof-Number (PN) search and its two-level variant PN^2 are explained thereafter. The chapter concludes with the experiments that resulted in solving Fanorona and its board variants. This chapter answers the second research question.

Next, in Chapter 5, expectimax and its pruning techniques, Star1 and Star2, are explained. We give an introduction to forward pruning and propose a new forward-pruning method for chance nodes, ChanceProbCut. We describe three two-player test domains, the games of Stratego (with imperfect information), Dice (non-deterministic) and ChanceBreakthrough (non-deterministic) and finish with experiments in each of these games. Chapter 5 gives an answer to the third research question.

In Chapter 6 we first give an introduction to the two main algorithms for deterministic multi-player games with perfect information, $\max^n$ and paranoid. We present a new algorithm for deterministic multi-player games with perfect information, called Best-Reply Search (BRS). Thereafter, Chinese Checkers, Focus, and Rolit are introduced as test domains. Next, experiments and a discussion are given. This chapter answers the fourth research question.

Chapter 7 summarizes the answers to the research questions and answers the problem statement. We also give directions for future research.

Chapter 2

Search Methods

This chapter describes basic search methods for turn-based games. We focus on the $\alpha\beta$ framework and Monte-Carlo Tree Search. These methods form the basis for the chapters that will follow. First, Section 2.1 defines several notions and concepts which we use throughout this thesis. Section 2.2 explains the minimax algorithm. The popular pruning mechanism $\alpha\beta$ is discussed in Section 2.3. The success of $\alpha\beta$ is strongly dependent on the move ordering (Marsland, 1986). The most prominent move-ordering techniques are explained in Section 2.4. Section 2.5 describes iterative deepening. Transposition tables are discussed in Section 2.6. Finally, Section 2.7 describes Monte-Carlo Tree Search and its enhancements.

2.1 Searching in Games

Russell and Norvig (2003) define a *game* as a search problem with the following components:

- The initial position; it includes the board configuration and an indication whose move it is.
- A set of operators, which define the legal moves that a player can make.
- A terminal test, that determines whether the game is over.
- An evaluation function, returning a value for the outcome of a game.

A *game tree* represents the state space of a game. A *node* in the tree represents a position in the game and an *edge* represents a legal move. A sequence of edges in the game tree forms a *path* if each edge has a node in common with the preceding and succeeding edge. The *root* of the tree is a representation of the initial position. A *terminal position* is a position where the game has ended (a win, a draw, or a loss). A *terminal node* represents a terminal position. A node is *expanded* when all successors are generated according to the game rules. Nodes on the path between the root and a node N are *ancestors* of N. This implies that N is a *descendant* of node M, if M is an ancestor. The nearest ancestor and descendants of a node are

called *parent* and *children*, respectively. Nodes having the same parent are *siblings*. A node is *interior* if it has at least one child.

A game tree is created by expanding all interior nodes. The game tree for the initial position is an explicit representation of all possible games which may be played (Pearl, 1984). The *game-theoretic value* is the value of the root when all participants play optimally. A *minimal game tree* is the minimal part of the game tree required to determine the game-theoretic value. Because for most games the (minimal) game tree is extremely large, determining the game-theoretic value is computationally not feasible.

The *search tree* is that part of the game tree which is analyzed. This tree has the same root but does not contain all nodes of the game tree. Nodes are generated during the *search process*. Nodes that do not have children (yet) are *leaf nodes*. A *subtree* of a tree is a node with all its descendants. The depth of a tree is the largest depth of all its leaves, counted in *plies*. A ply corresponds to a turn of a player (Samuel, 1959). The *search depth* of a node is the number of plies which still need to be searched at this node.

2.2 Minimax

For two-player games, the players are called MAX and MIN. The MAX player tries to maximize the evaluation function while the MIN player tries to minimize it.

The *minimax* algorithm (Von Neumann and Morgenstern, 1944) is designed to find the optimal move for the MAX player, whose turn it is. Every move in the initial position is expanded in a recursive depth-first way until the end of the game is reached, creating a search tree. The evaluation function is used to assign game-theoretic values to each outcome. One step at a time, these values are backed up to the root, where MAX always chooses the highest value and MIN always chooses the lowest value. When all moves have been investigated at the root, the move with highest value is played.

An example of a minimax tree is depicted in Figure 2.1. In this case, a game of Tic-Tac-Toe is played. It is the X player's turn at the root, which takes the role of the MAX player. The moves are explored to the end of the game where the evaluation function is used to assign a value to a terminal position. The values for loss, draw, and win are -1, 0, and 1, respectively. The numbers along the edges indicate in which order nodes are explored.

For non-trivial games, it is usually not possible within a limited amount of time to traverse the complete game tree until terminal positions are reached. Therefore, the game tree is searched until a fixed depth, where a *heuristic* evaluation function indicates the desirability of the position at the leaf node.

Instead of having two separate MAX and MIN methods, it is possible to use a single method, saving a significant number of lines of code. At every ply, the values of the children are negated. This method is called *negamax* (Knuth and Moore, 1975). The pseudo code for negamax can be found in Algorithm 2.1. The variable *turn* is used for distinguishing MAX and MIN nodes, and can have value 1 for a MAX node, and -1 for a MIN node.

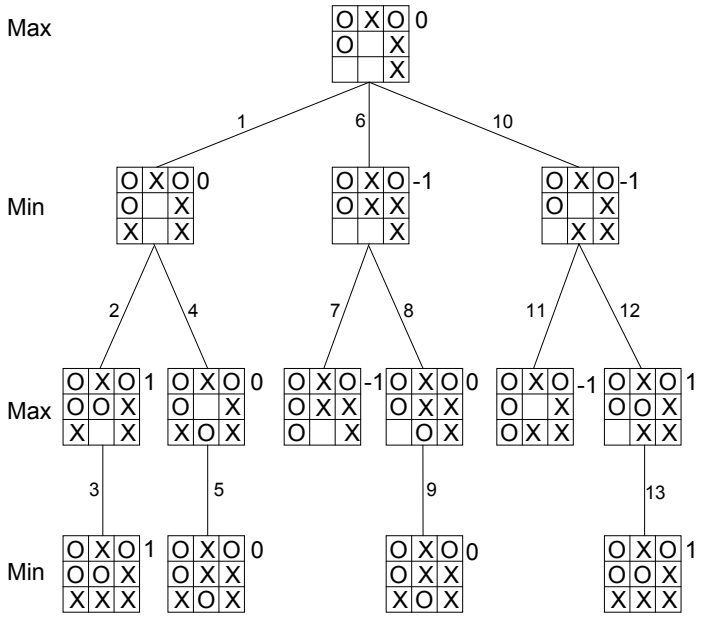


Figure 2.1: An example minimax tree.

```
 1: negamax(position, depth, turn)
 2:
 3: if terminal(position) or depth==0 then
 4:    return turn × eval();
 5: end if
 6:
 7: best = −∞;
 8: for all Moves i do
 9:    doMove(position, i);
10:    best = max(best, −negamax(position, depth−1, −turn));
11:    undoMove(position, i);
12: end for
13:
14: return best;
```

Algorithm 2.1: Negamax pseudo code.

2.3 $\alpha\beta$ Search

It is not necessary to investigate every node of the search tree to determine the minimax value of the root. The process of eliminating branches of the search tree from consideration without examining them is called *pruning*. The most-famous pruning method is $\alpha\beta$ pruning (Knuth and Moore, 1975).

$\alpha\beta$ produces a cutoff in a node if a returned value indicates that the remaining children cannot alter the value of the root. To prune the remaining children, the returned value has to be greater than or equal to the value of its ancestors of the same type (MAX or MIN). Knuth and Moore (1975) proved that in the best case $O(b^{d/2})$ nodes will be expanded, where b is the average branching factor, and d is the search depth.

The pseudo code for $\alpha\beta$ in the negamax framework is shown in Algorithm 2.2. Only small modifications of the minimax pseudo code (Algorithm 2.1) are required. At the root node, α and β are initialized to $-\infty$ and ∞, respectively.

```
 1: alphabeta(position, depth, turn, alpha, beta)
 2:
 3: if terminal(position) or depth==0 then
 4:   return turn × eval();
 5: end if
 6:
 7: for all Moves i do
 8:   doMove(position, i);
 9:   alpha = max(alpha, −alphabeta(position, depth−1, −turn, −beta, −alpha));
10:   undoMove(position, i);
11:   if alpha ≥ beta then
12:     return beta;
13:   end if
14: end for
15:
16: return alpha;
```

Algorithm 2.2: $\alpha\beta$ pseudo code.

Figure 2.2 depicts an example of $\alpha\beta$ pruning (the initial position is identical to the example in Figure 2.1). In this tree, both players have an optimal move ordering, i.e., the strongest move is always investigated first. $\alpha\beta$ is able to prune twice in this example. After node A has been searched, the value of the root is ≥ 0, because the other two moves may have a higher value. After edge 7 has been investigated, the value of the node B is ≤ -1 (actually equals -1 because it is the smallest possible value). The MAX player now always prefers A above B. Therefore, all other moves under B are irrelevant to the value of the root and do not require to be searched anymore. With the same reasoning, the right subtree of C can be pruned.

We remark that three types of nodes can be distinguished in $\alpha\beta$ search (Knuth and Moore, 1975; Marsland and Popowich, 1985). (1) *PV nodes* form the *principal variation* of the search (i.e., the expected line of play). At a PV node, all the children have to be investigated. The best move found in a PV node leads to a successor PV node, while all other investigated children are CUT nodes. (2) At *CUT nodes* a cutoff takes place. Ideally, only one child has to be explored. At a CUT node the child causing the cutoff is an ALL node. (3) In an *ALL node* all children have to be explored. The successors of an ALL node are CUT nodes.

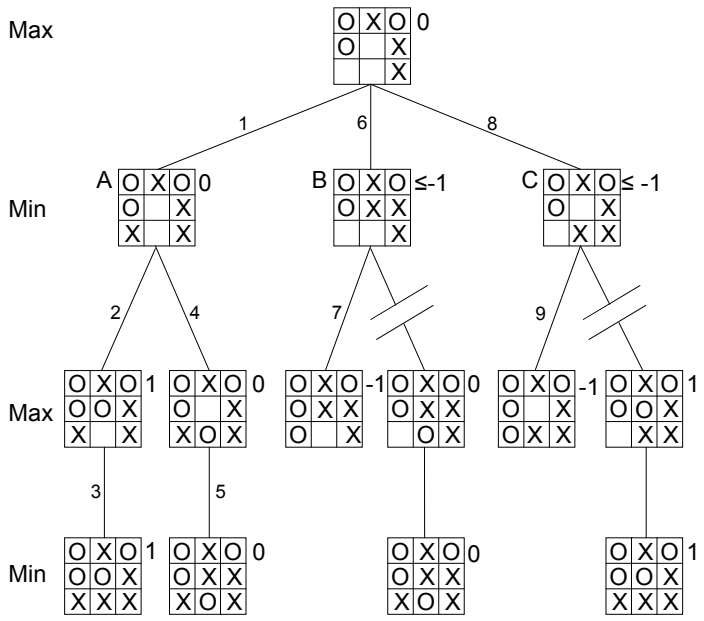


Figure 2.2: An example $\alpha\beta$ tree.

2.4 Move Ordering

The success of $\alpha\beta$ search is strongly dependent on the move ordering (Marsland, 1986). Therefore, researchers have investigated methods to examine the strongest move first. We distinguish two types of move ordering. (1) *Static* move ordering is independent of the search. These techniques rely on domain knowledge (e.g., capturing a queen in chess) or by learning techniques (e.g., the Neural MoveMap Heuristic, Kocsis, Uiterwijk, and Van den Herik, 2001). (2) *Dynamic* move ordering is dependent on the search. These techniques rely on information gained during the search. We describe two dynamic move-ordering techniques in the following subsections. Subsection 2.4.1 gives an introduction to the killer heuristic. In Subsection 2.4.2 the history heuristic is explained.

2.4.1 Killer Heuristic

The basic assumption of the *killer heuristic* (Akl and Newborn, 1977) is that a move which produces a cutoff in one position, often produces a cutoff in a similar position, if the move is legal. The killer heuristic stores for every ply at least one killer move which has produced a pruning before at a node in the corresponding ply. When a new node is searched, the killer moves for that ply are examined first if they are legal. A cutoff may occur even before all possible moves are generated. When a move produces a pruning which is not a killer move for that ply, it becomes a new

killer move, overwriting an old one. The killer move selected for deletion, is the one that has not been used for the longest time.

2.4.2 History Heuristic

The *history heuristic* is a dynamic move-ordering technique based on the number of cutoffs caused by a move, irrespective of the position in which the move has been made. It was proposed by Schaeffer (1983) and has been adopted in several game-playing programs.

For every move seen in the game tree a history is maintained. Because there is only a limited number of legal moves, it is possible to maintain a score for each move in n tables (n is the number of players). Moves are typically indexed by their coordinates on the board, independent of how the rest of the board looks like (e.g., 64 from squares $\times$ 64 to squares for chess). When an interior node is searched, the history-table entry for the best move found is incremented by some value (e.g., 2^d, where d is the search depth of the subtree under the node). This move either caused an $\alpha\beta$ cutoff or had the best score after all moves had been searched.

When in a node all possible moves have been generated, they are ordered according to their history-heuristic scores. This may lead to more $\alpha\beta$ cutoffs. The scores in the tables can be maintained during the whole game. Each time a new search is started, the scores are decremented by a factor (e.g., divided by 2). They are only reset to zero or to some default values at the beginning of a complete new game. The two drawbacks of the history heuristic are (1) that it assumes that all moves occur equally often and (2) for illegal moves memory is reserved in the tables (Hartmann, 1988).

There exist three variations on the history heuristic. (1) To counter somewhat the two disadvantages Hartmann (1988) proposed an alternative for the history heuristic, the *butterfly heuristic*. This heuristic takes the move frequencies in the search trees into account. Instead of history tables, *butterfly boards* are used. They are defined in the same way as in the history heuristic. Any move that is not cut is recorded. Each time a move is executed in the search tree, its corresponding entry in the butterfly board is also incremented by a value. (2) The countermove heuristic (Uiterwijk, 1992) is based on the assumption that many moves have a "natural" response irrespective of the actual position in which the moves occur. (3) Winands *et al.* (2006) proposed the *relative history heuristic* which divides the history-heuristic score by the butterfly-heuristic score. Their experiments reveal that it reduces the number of nodes searched in the games of Lines of Action (LOA) and Go even more.

2.5 Iterative Deepening

Usually $\alpha\beta$ investigates the search tree up to a predefined depth. However, it is not straightforward to predict the running time. *Iterative deepening* overcomes this problem by gradually increasing the search depth, typically by one ply per iteration. This is done until time runs out. Although this may seem inefficient, the overhead is rather small. We provide the following example. To finish a search of depth d and

average branching factor b, b^d nodes are required. When using iterative deepening, the root is expanded d times, the 1-ply deep nodes $d-1$ times, up to the d-ply deep nodes, which are only expanded once. Therefore, iterative deepening expands $\sum_{i=0}^{d}(d+1-i)\times b^i$ nodes. With an average branching factor of 10 and search depth 5, the regular search expands $10^5 = 100,000$ nodes and iterative deepening expands $6+50+400+3,000+20,000+100,000 = 123,456$ nodes. The larger the branching factor, the smaller the overhead of repeatedly expanded nodes. In spite of this overhead, iterative deepening is able to reduce the number of nodes searched (Slate and Atkin, 1977). The reason for this is that information from previous iterations is used by the killer and history heuristic and is re-used to improve the quality of the move ordering at the next iteration. Also the transposition table benefits from iterative deepening, which is introduced in the next section.

2.6 Transpositions

In many games, there exist multiple paths to the same position. In these games, it is possible to reduce search effort by storing earlier search results. If an identical position is encountered during the search, it is called a *transposition*. This implies that the search tree can be considered a *graph*, because of transpositions. In a transposition, the information of the previous search may be reused to reduce search effort. While the position may be identical, this history of the game leading to the position is not. For some games (e.g., chess), a three-times repetition draw rule makes the identity of a position dependent on the history (Slate and Atkin, 1977). This is known as the *graph-history-interaction* (GHI) problem (Palay, 1983; Campbell, 1985).

Transposition tables are explained in Subsection 2.6.1. Subsection 2.6.2 describes Enhanced Transposition Cutoffs.

2.6.1 Transposition Tables

When a position has been investigated, positional information is stored in a *transposition table* (Greenblatt, Eastlake, and Crocker, 1967). Because memory limitations do not allow every position to be stored, the transposition table is implemented as a finite *hash table* (Knuth, 1973). A *hash value* is computed for the position using some hashing method, such as Zobrist (1970) hashing. If the transposition table consists of 2^n entries, the n low-order bits of the hash value are used as a *hash index*. The remaining bits (the *hash key*) are used to distinguish between positions with the same hash index. The size of the hash index should be chosen sufficiently large (Hyatt, Grover, and Nelson, 1990).

A typical entry consists of the following five components. (1) The hash key. (2) The best move for this position. This move either obtained the highest score or produced a cutoff. (3) The score of the best move. In $\alpha\beta$ search, this may be an exact value, an upper or a lower bound. (4) A *flag* that indicates whether the score is an exact value, upper or lower bound. (5) A search depth that indicates how deep this position has been investigated.

When a transposition is encountered, the information in the transposition table may be used in three ways, depending on the depth and flag stored in the entry. Let the depth stored in the transposition table be ttd and the depth still to be searched d. (1) If $ttd \geq d$ and an exact value is available, the search is aborted and the transposition table score is returned. (2) If $ttd \geq d$ and the score is not exact, but an upper or a lower bound, the current $\alpha\beta$ window is adapted accordingly and the stored move is used for move ordering. (3) If $ttd < d$, the stored move is used for move ordering.

If two different positions have the same hash value, a so-called *type-1 error* occurs. This is a serious error which is difficult to detect. Let N be the number of distinguishable positions and M be the number of different positions which have to be stored in the transposition table. The probability that all M positions will have different hash values is given by

$$P\left(no\ \ errors\right) \approx e^{-\frac{M^2}{2N}} \tag{2.1}$$

If the size of the transposition table is sufficiently large, the probability of a type-1 error is negligible (Breuker, 1998).

If two different positions have the same hash index, but different hash key, a so-called *type-2 error* occurs. This is also known as *collision* (Knuth and Moore, 1975). When a collision occurs, one has to choose which position to store. This choice is based on a *replacement scheme* (Breuker, Uiterwijk, and Van den Herik, 1994; Breuker, Uiterwijk, and Van den Herik, 1996). For an overview of replacement schemes, we refer to Beal and Smith (1996) and Breuker (1998). The most commonly used scheme is the two-deep replacement scheme (Breuker, 1998).

2.6.2 Enhanced Transposition Cutoff

Enhanced Transposition Cutoff (ETC) (Schaeffer and Plaat, 1996) is a method to produce a cutoff using the transposition table, even though an entry does not exist for the current position. Children of the current position may be stored in the transposition table and one of these may produce a cutoff without further expanding the current node. After the moves of the current positions have been created, all children are queried in the transposition table. If a child is stored in the table, the α value of the parent may be updated, possibly resulting in a cutoff. Because calculating a hash value for a position and examining in the transposition table takes time, ETC is only applied if the search depth is large enough.

2.7 Monte-Carlo Tree Search

Recently, Monte-Carlo (MC) methods have become a popular approach for intelligent play in games. MC simulations have first been used as an evaluation function inside a classic search tree (Brügmann, 1993; Bouzy and Helmstetter, 2003). In this role, MC simulations have been applied to Backgammon (Tesauro and Galperin, 1997), Poker (Billings *et al.*, 1999), Scrabble (Sheppard, 2002b), Morpion Solitaire

and Gaps (Helmstetter, 2007). Due to the expensive evaluation, the search is not able to investigate the search tree sufficiently deep in some games (Chaslot, 2010).

Therefore, the MC simulations have been placed into a tree-search context in multiple ways (Chaslot *et al.*, 2006b; Coulom, 2007a; Kocsis and Szepesvári, 2006). The resulting general method is called Monte-Carlo Tree Search (MCTS) (Coulom, 2007a; Kocsis and Szepesvári, 2006). MCTS is a best-first search method guided by Monte-Carlo simulations. In contrast to classic tree-search algorithms such as $\alpha\beta$ (Knuth and Moore, 1975) and A* (Hart *et al.*, 1968), MCTS does not require a heuristic evaluation function. MCTS is particularly interesting for domains where building an evaluation function is a difficult or time-consuming task, such as the game of Go. MCTS builds a search tree employing MC simulations at the leaf nodes. Each node in the tree represents a position and typically stores the average score of the simulations played through this node, and the number of visits. MCTS constitutes a family of tree-search algorithms not only applicable to games, but also to scheduling and optimization problems (cf. Chaslot *et al.*, 2006a; Mesmay *et al.*, 2009)

Subsection 2.7.1 describes the structure of MCTS. Thereafter, Subsection 2.7.2 explains three enhancements for MCTS: Progressive Bias, Progressive Widening and RAVE. Finally, Subsection 2.7.3 discusses how MCTS may be parallelized.

2.7.1 Structure of MCTS

MCTS consists of two parts: a relatively shallow search tree and deep simulated games. The tree structure determines the first moves of the simulated games. The results of these simulated games shape the tree. In general, MCTS consists of four steps (Chaslot *et al.*, 2008d). These steps are visualized in Figure 2.3. (1) During the selection step the search tree is traversed starting from the root node until a position is encountered which is not stored in the tree. (2) Subsequently, during the play-out step moves are played until the end of the game. (3) Next, in the expansion step a number of nodes is added to the tree. (4) Finally, in the backpropagation step the result of a simulated game is propagated backwards, through the previously traversed nodes. We discuss strategies to perform the four basic steps of MCTS in the remainder of this subsection.

The four steps are iterated until the time runs out. When this happens, a final move selection is used to determine which move should be played in the actual game.

Selection Step

From the root node, a *selection strategy* is applied recursively until a position is reached that is not a part of the tree yet. The selection strategy controls the balance between *exploitation* and *exploration* (Chaslot, 2010). On the one hand, the moves that lead to the best results so far are selected (exploitation). On the other hand, the less promising moves still must be tried, due to the uncertainty of the evaluation (exploration).

Several selection strategies have been designed for MCTS. Kocsis and Szepesvári (2006) proposed the selection strategy UCT (Upper Confidence bounds applied to

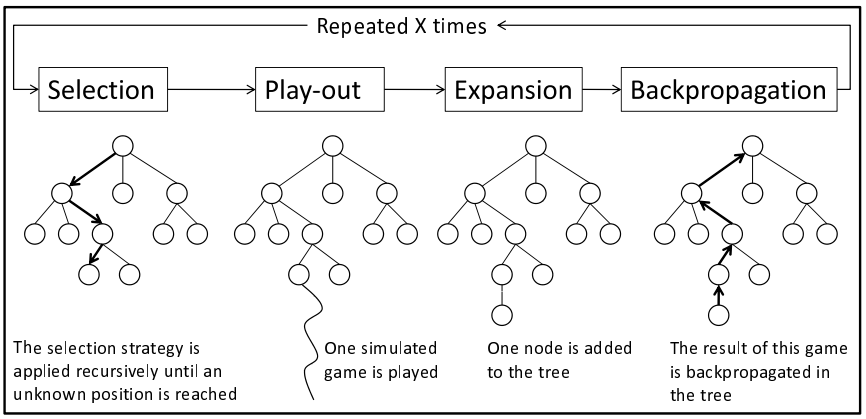


Figure 2.3: The four steps of MCTS (slightly adapted from Chaslot *et al.*, 2008d).

Trees). This strategy is straightforward to implement, and used in many programs. At node p with children i, the child is chosen that maximizes the following formula.

$$v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} \tag{2.2}$$

v_i is the value of the child i, n_i is the visit count of i, and n_p is the visit count of node p. C is a coefficient, which has to be tuned experimentally. Other selection strategies are OMC (Chaslot *et al.*, 2006b), UCB1-TUNED (Gelly and Wang, 2006), PBBM (Coulom, 2007a) and MOSS (Audibert and Bubeck, 2009).

Here we remark that Coulom (2007a) chooses a move according to the selection strategy only if n_p is above a certain threshold. Before the threshold is reached, the simulation strategy is used. The latter is explained below.

Play-Out Step

The play-out step begins when the search enters a position that is not part of the tree yet. Moves are then played until the end of the game. These moves are chosen according to a *simulation strategy*. A completed game is called a *play-out*. The play-outs are an estimate for the values of the nodes in the MCTS tree. The use of an adequate simulation strategy (instead of playing randomly) has been shown to improve the level of play significantly (Bouzy, 2005; Gelly *et al.*, 2006; Chen and Zhang, 2008).

The moves may be chosen quasi-randomly based on heuristic knowledge (Gelly and Silver, 2007). A simulation strategy is subject to two tradeoffs (Chaslot, 2010). The first tradeoff is between search and knowledge. While knowledge increases the playing strength, it decreases the simulation speed. The second tradeoff deals with exploration and exploitation. If the strategy is too explorative, the search tree is too shallow and if the strategy is too exploitative, the search will become too selective. Ideally, the used heuristic knowledge should be fast to compute, increase the quality

of the play-out significantly and at the same time allow enough randomness for exploration.

Expansion Step

Usually the whole game tree cannot be stored in memory. An *expansion strategy* decides whether a node is expanded by storing one or more of its children in memory. Coulom (2007a) proposed to store the first encountered position that was not stored in the tree, expanding one node per simulation.

Backpropagation Step

During the backpropagation step, the result of the simulation at a leaf node is propagated backwards to the root. A *backpropagation strategy* decides how the value of a node is used to change the values of its ancestors.

There exist several backpropagation strategies. The most-applied one is Average, which takes the average of all simulations made through that node (Coulom, 2007a). The Max strategy (Knuth and Moore, 1975) backpropagates values in a negamax way, but was proven to be too noisy (Chaslot *et al.*, 2006b; Coulom, 2007a). The Informed-Average strategy (Chaslot *et al.*, 2006b) aims to converge faster to the value of the best move than Average by assigning a bigger weight to the best moves. Coulom (2007a) proposed the Mix strategy, which initially resembles the Average strategy, but converges to the Max strategy. The MCTS-Solver strategy (Winands, Björnsson, and Saito, 2008) also backpropagates game-theoretic values, allowing MCTS to play narrow tactical lines better in sudden-death games.

2.7.2 Enhancements

This subsection introduces three well-known enhancements for MCTS. First, progressive bias is introduced. Second, progressive widening is discussed. Finally, an introduction to RAVE is given.

Progressive Bias

The aim of the *progressive-bias* strategy is to direct the search according to heuristic knowledge (Chaslot *et al.*, 2008d). For that purpose, the selection strategy is modified according to that knowledge. The influence of this modification is important when a few games have been played, but decreases with a growing number of games. The UCT formula (Equation 2.2) is adapted in the following way.

$$v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + \frac{H_i}{n_i + 1} \tag{2.3}$$

where H_i represents heuristic knowledge, which depends only on the board configuration represented by the node i. When n_i is small, the heuristic factor is most dominant. With increasing n_i, a balance is found between the results of the simulated games and the heuristic knowledge. When n_i is large, the results of the simulated games are the dominant factor.

Progressive Widening

The heuristic score H_i, used by progressive bias, may also be used for a method called *progressive widening* (Coulom, 2007b; Chaslot *et al.*, 2008d). When the number of simulations is small, the simulation strategy is used as selection strategy. Progressive widening reduces the branching factor in this phase based on the heuristic knowledge H_i and gradually increases the branching factor when more simulations are performed.

Rapid Action-Value Estimation

Brügmann (1993) proposed to acquire results from simulations quicker by the *all-move-as-first heuristic* (AMAF). AMAF assigns the result of a simulated game not only to the first move played, but to all moves during the game. $\text{AMAF}_{s,m}$ is the AMAF value for move m in position s. Gelly and Silver (2007) proposed a method called *Rapid Action-Value Estimation* (RAVE), that uses the AMAF value in combination with MCTS. The UCT formula (Formula 2.2) is adapted in the following way.

$$(1 - \beta(n_p)) \times (v_i + C \times \sqrt{\frac{\ln n_p}{n_i}}) + \beta(n_p) \times AMAF_{p,i} \tag{2.4}$$

Gelly and Silver (2007) proposed to use $\beta(N) = \sqrt{\frac{k}{3N+k}}$ with k subject to tuning, which has led to good results. RAVE was successfully applied in Go (Gelly and Silver, 2007) and Havannah (Teytaud and Teytaud, 2010). In games where pieces are continually moving across the board, such as Chinese Checkers, it is more difficult for RAVE to work effectively (Sturtevant, 2008b).

2.7.3 Parallelization

Just as for $\alpha\beta$ search, it holds for MCTS that the more time is spent for selecting a move, the better the game play is. The recent developments in hardware have gone into the direction that nowadays even personal computers contain several cores. To get the most out of the available time, one has to parallelize AI techniques to use all available hardware (Chaslot, 2010).

Cazenave and Jouandeau (2007) proposed two parallelization methods, leaf and root parallelization, which will be introduced first. Thereafter, tree parallelization (Chaslot, Winands, and Van den Herik, 2008b) will be described.

Leaf Parallelization

Leaf parallelization (Cazenave and Jouandeau, 2007) is a straightforward way to parallelize MCTS. One thread traverses the tree just as in regular MCTS. Next, starting from the leaf node, one play-out is performed for each available thread. When all games are finished, the results of all these play-outs are propagated backwards through the tree as usual.

Leaf parallelization has two problems. (1) Play-outs have a variable length, and the search has to wait until the longest game has finished. (2) Information between the play-outs is not shared. When the first play-outs indicate that the node most likely leads to a loss, the remaining play-outs may be a waste of computational time.

Root Parallelization

Root parallelization (Cazenave and Jouandeau, 2007) consists of building multiple MCTS trees in parallel, with one thread per tree. The threads do not share information with each other. Each individual thread performs a regular MCTS. When the available time is spent, the separate trees are combined and the choice of which move to play is based on all results. Root parallelization is a straightforward and efficient way to parallelize MCTS. Chaslot *et al.* (2008b) show that root parallelization nearly has a linear speedup for a small number of threads.

Tree Parallelization

Tree parallelization (Chaslot *et al.*, 2008b) is a parallelization method in which threads share information with each other. This method uses one shared tree from which several simultaneous play-outs are played. Each thread can modify the information contained in the tree. Mutexes are required to lock certain parts of the tree to prevent data corruption. Based on the location of the mutexes in the tree, we distinguish two mutex-location methods: (1) using a *global mutex* and (2) using several *local mutexes*. With a global mutex the complete tree is locked when a thread edits information. With a local mutex only the node is locked that requires updating. This makes it possible that multiple threads edit the tree simultaneously, decreasing waiting time. Enzenberger and Müller (2010) showed that a mutex-free tree parallelization outperformed a global-mutex tree parallelization for the Go program Fuego. To prevent that all threads select the same node in the tree, a *virtual loss* may be assigned to a node if a thread selects it (Chaslot *et al.*, 2008b). The virtual loss is removed when the thread that gave the virtual loss starts backpropagating the result of the play-out.

Chapter 3

Single-Player Monte-Carlo Tree Search

This chapter is an updated and abridged version of the following publications:

1. Schadd, M.P.D., Winands, M.H.M., Herik, Chaslot, G.M.J-B., H.J. van den, and Uiterwijk, J.W.H.M. (2008a). Single-Player Monte-Carlo Tree Search. *Proceedings of the 20st BeNeLux Conference on Artificial Intelligence (BNAIC'08)* (eds. A. Nijholt, M. Pantic, M. Poel, and H. Hondorp), pp. 361–362, University of Twente, Enschede, The Netherlands.
2. Schadd, M.P.D., Winands, M.H.M., Herik, H.J. van den, and Aldewereld, H. (2008b). Addressing NP-Complete Puzzles with Monte-Carlo Methods. *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning*, Vol. 9, pp. 55–61, The Society for the Study of Artificial Intelligence and Simulation of Behaviour, Brighton, United Kingdom.
3. Schadd, M.P.D., Winands, M.H.M., Herik, H.J. van den, Chaslot, G.M.J-B. and Uiterwijk, J.W.H.M. (2008c). Single-Player Monte-Carlo Tree Search. *Computers and Games (CG 2008)* (eds. H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 1–12, Springer-Verlag, Berlin, Germany.

The traditional approaches to deterministic one-player games with perfect information (Kendall, Parkes, and Spoerer, 2008) are applying A* (Hart *et al.*, 1968) or IDA* (Korf, 1985). These methods have been quite successful for solving this type of games. The disadvantage of the methods is that they require an admissible heuristic evaluation function. The construction of such a function can be difficult. Since Monte-Carlo Tree Search (MCTS) does not require an admissible heuristic, it may be an interesting alternative. Because of its success in two-player games (cf. Lee, Müller, and Teytaud, 2010) and multi-player games (Sturtevant, 2008a), this chapter investigates the application of MCTS in deterministic one-player games with perfect information.

So far, MCTS has not been widely applied in one-player games. One example is the Sailing Domain (Kocsis and Szepesvári, 2006), which is a non-deterministic game with perfect information. MCTS has also been used for optimization and planning problems which can be represented as deterministic one-player games. Chaslot *et al.* (2006a) applied MCTS in production management problems. Mesmay *et al.* (2009) proposed the MCTS variant TAG for optimizing libraries for different platforms. Schadd *et al.* (2008c) showed that MCTS was able to achieve high scores in the puzzle[1] SameGame.

This chapter answers the first research question by proposing an MCTS method for a one-player game, called Single-Player Monte-Carlo Tree Search (SP-MCTS). MCTS for two-player games, as described in Section 2.7, forms the starting point for this search method. We adapted MCTS by two modifications resulting in SP-MCTS. The modifications are (1) in the selection strategy and (2) in the backpropagation strategy. SP-MCTS is tested in the game of SameGame, because there exists no reliable admissible heuristic evaluation function for this game.

The article is organized as follows. In Section 3.1 we present the rules, complexity and related work of SameGame. In Section 3.2 we discuss why the classic approaches A* and IDA* are not suitable for SameGame. Then, we introduce the SP-MCTS approach in Section 3.3. Section 3.4 describes the Cross-Entropy Method which is used for tuning the SP-MCTS parameters. Experiments and results are given in Section 3.5. Section 3.6 gives the chapter conclusions and indicates future research.

3.1 SameGame

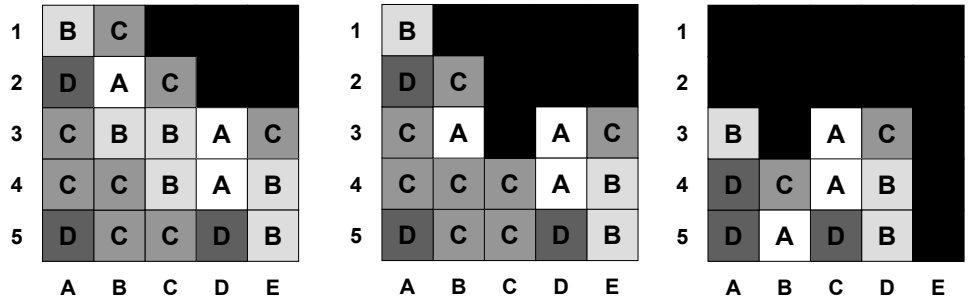
SameGame is a puzzle invented by Kuniaki Moribe under the name *Chain Shot!* in 1985. It was distributed for Fujitsu FM-8/7 series in a monthly personal computer magazine called *Gekkan ASCII* (Moribe, 1985). The puzzle was afterwards re-created by Eiji Fukumoto under the name of *SameGame* in 1992.

In this section, we first explain the rules in Subsection 3.1.1. Subsequently, we give an analysis of the complexity of SameGame in Subsection 3.1.2. Finally, we present related work in Subsection 3.1.3.

3.1.1 Rules

SameGame is played on a vertically oriented 15×15 board initially filled with blocks of 5 colors at random. A move consists of removing a group of (at least two) orthogonally adjacent blocks of the same color. The blocks on top of the removed group fall down. As soon as an empty column occurs, the columns to the right of the empty column are shifted to the left. Therefore, it is impossible to create separate subgames. For each removed group points are rewarded. The number of points is dependent on the number of blocks removed and can be computed by the formula $(n-2)^2$, where n is the size of the removed group.

[1] From now on, we call one-player deterministic games with perfect information for the sake of brevity *puzzles* (Kendall *et al.*, 2008).



(a) Playing "B" in the center column.

(b) Playing "C" in the center column.

(c) Resulting position.

Figure 3.1: Example SameGame moves.

We show two example moves in Figure 3.1. When the "B" group in the third column with a connection to the second column of position 3.1(a) is played, the "B" group is removed from the game. In the second column the "CA" blocks fall down and in the third column the "C" block falls down, resulting in position 3.1(b). Due to this move, it is now possible to remove a large group of "C" blocks ($n = 6$). Owing to an empty column the two columns at the right side of the board are shifted to the left, resulting in position 3.1(c).[2] The first move is worth 1 point; the second move is worth 16 points.

The game is over if no more blocks can be removed. This happens when either the player (1) has removed all blocks or (2) is left with a position where no adjacent blocks have the same color. In the first case, 1,000 bonus points are rewarded. In the second case, points are deducted. The formula for deducting is similar to the formula for awarding points but now iteratively applied for each color left on the board. Here it is assumed that all blocks of the same color are connected.

There are variations that differ in board size and the number of colors, but the 15×15 variant with 5 colors is the accepted standard. If a variant differs in the scoring function, it is named differently (e.g., Clickomania or Jawbreaker, cf. Biedl *et al.*, 2002; Julien, 2008).

3.1.2 Complexity of SameGame

The complexity of a game indicates a measure of difficulty for solving the game. Two important measures for the complexity of a game are the game-tree complexity and the state-space complexity (Allis, 1994). The game-tree complexity is an estimation of the number of leaf nodes that the complete search tree would contain to solve the initial position. The state-space complexity indicates the total number of possible states.

For SameGame these complexities are as follows. The game-tree complexity

[2] Shifting the columns at the left side to the right would not have made a difference in number of points. For consistency, we always shift columns to the left.

can be approximated by simulation. By randomly playing 10^6 puzzles, the average length of the game was estimated to be 64.4 moves and the average branching factor to be 20.7, resulting in a game-tree complexity of 10^{85}. The state-space complexity is computed rather straightforwardly. It is possible to calculate the number of combinations for one column by $C = \sum_{n=0}^{r} c^n$ where r is the height of the column and c is the number of colors. To compute the state-space complexity we take C^k where k is the number of columns. For SameGame there exist 10^{159} states. This is an over-estimation because a small percentage of the positions are symmetrical.

Furthermore, the difficulty of a game can be described by deciding to which complexity class it belongs (Johnson, 1990). The similar game Clickomania was proven to be NP-complete by Biedl *et al.* (2002). However, the complexity of SameGame could be different. The more points are rewarded for removing large groups, the more the characteristics of the game may differ from Clickomania. In Clickomania the only goal is to remove as many blocks as possible, whereas in SameGame points are rewarded for removing large groups as well.

Theorem. SameGame is at least as difficult as Clickomania.

Proof. A solution S of a SameGame problem is defined as a path from the initial position to a terminal position. Either S (1) has removed all blocks from the game or (2) has finished with blocks remaining on the board. In both cases a search has to be performed to investigate whether a solution exists that improves the score and clears the board.

Clickomania is a variant of SameGame where no points are rewarded and the only objective is to clear the board. Finding only one solution to this problem is easier than finding the highest-scoring solution (as in SameGame). Therefore, SameGame is at least as difficult as Clickomania. □

3.1.3 Related Work

For the game of SameGame some research has been performed. The contributions are benchmarked on a standardized test set of 20 positions.[3] The first SameGame program has been written by Billings (2007). This program applies a non-documented method called *Depth-Budgeted Search* (DBS). When the search reaches a depth where its budget has been spent, a greedy simulation is performed. On the test set his program achieved a total score of 72,816 points with 2 to 3 hours computing time per position. Schadd *et al.* (2008c) set a new high score of 73,998 points by using Single-Player Monte-Carlo Tree Search (SP-MCTS). This chapter will describe SP-MCTS in detail. Takes and Kosters (2009) proposed *Monte Carlo with Roulette-Wheel Selection* (MC-RWS). It is a simulation strategy that tries to maximize the size of one group of a certain color and at the same time tries to create larger groups of another color. On the test set their program achieved a total score of 76,764 points with a time limit of 2 hours. In the same year Cazenave (2009) applied Nested Monte-Carlo Search which led to an even higher score of 77,934. Until the year 2010, the top score on this set was 84,414 points, held by the program

[3]The positions can be found at: www.js-games.de/eng/games/samegame.

SPURIOUS AI.[4] This program applies a method called Simple Breadth Search (SBS), which uses beam search, multiple processors and a large amount of memory (cf. Takes and Kosters, 2009). Further details about this program are not known. Later in 2010 this record was claimed to be broken with 84,718 points by using a method called *Heuristically Guided Swarm Tree Search* (HGSTS) (Edelkamp *et al.*, 2010), which is a parallelized version of MCTS.

3.2 A* and IDA*

The classic approach to puzzles involves methods such as A* (Hart *et al.*, 1968) and IDA* (Korf, 1985). A* is a best-first search where all nodes have to be stored in a list. The list is sorted by an admissible evaluation function. At each iteration the first element is removed from the list and its children are added to the sorted list. This process is continued until the goal state arrives at the start of the list.

IDA* is an iterative deepening variant of A* search. It uses a depth-first approach in such a way that there is no need to store the complete tree in memory. The search continues depth-first until the cost of arriving at a leaf node and the value of the evaluation function exceeds a certain threshold. When the search returns without a result, the threshold is increased.

Both methods are strongly dependent on the quality of the evaluation function. Even if the function is an admissible under-estimator, it still has to give an accurate estimation. Classic puzzles where this approach works well are the Eight Puzzle with its larger relatives (Korf, 1985; Sadikov and Bratko, 2007) and Sokoban (Junghanns, 1999). Here a good under-estimator is the well-known Manhattan Distance. The main task in this field of research is to improve the evaluation function, e.g., with pattern databases (Culberson and Schaeffer, 1998; Felner *et al.*, 2005).

These classic methods fail for SameGame because it is not straightforward to make an admissible function that still gives an accurate estimation. An attempt to make such an evaluation function is by just awarding points to the current groups on the board. This resembles the score of a game where all groups are removed in a top-down manner. However, if an optimal solution to a SameGame problem has to be found, we may argue that an "over-estimator" of the position is required, because in SameGame the score has to be maximized, whereas in common applications costs have to be minimized (e.g., shortest path to a goal). An admissible "over-estimator" can be created by assuming that all blocks of the same color are connected and would be able to be removed at once. This function can be improved by checking whether there is a color with only one block remaining on the board. If this is the case, the 1,000 bonus points for clearing the board may be deducted because the board cannot be cleared completely. However, such an evaluation function is far from the real score for a position and does not give good results with A* and IDA*. Our tests have shown that using A* and IDA* with the proposed "over-estimator" results in a kind of breadth-first search. The problem is that after expanding a node, the heuristic value of a child can be significantly lower than the value of its parent, unless a move removes all blocks with one color from the board. We expect that

[4]The exact date when the scores were uploaded to http://www.js-games.de/ is unknown.

other Depth-First Branch-and-Bound methods (Vempaty, Kumar, and Korf, 1991) suffer from the same problem. Since no good evaluation function has been found yet, SameGame presents a new challenge for puzzle research.

3.3 Single-Player Monte-Carlo Tree Search

Based on MCTS, we propose an adapted version for puzzles: Single-Player Monte-Carlo Tree Search (SP-MCTS). We discuss the four steps (selection, play-out, expansion and backpropagation) and point out differences between SP-MCTS and MCTS in Subsections 3.3.1-3.3.4. SameGame serves as example domain to explain SP-MCTS. The final move selection is described in Subsection 3.3.5. Subsection 3.3.6 describes how randomized restarts may improve the score.

3.3.1 Selection Step

Selection is the strategic task to select one of the children of a given node. It controls the balance between *exploitation* and *exploration*. Exploitation is the task to focus on the moves that led to the best results so far. Exploration deals with the less promising moves that still may have to be explored, due to the uncertainty of their evaluation so far. In MCTS at each node starting from the root, a child has to be selected until a position is reached that is not part of the tree yet. Several strategies have been designed for this task (Chaslot *et al.*, 2006b; Kocsis and Szepesvári, 2006; Coulom, 2007a).

Kocsis and Szepesvári (2006) proposed the selection strategy UCT (Upper Confidence bounds applied to Trees). For SP-MCTS, we use a modified UCT version. At the selection of node p with children i, the strategy chooses the move, which maximizes the following formula.

$$v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + \sqrt{\frac{\sum r^2 - n_i \times v_i^2 + D}{n_i}} \tag{3.1}$$

The first two terms constitute the original UCT formula. It uses n_i as the number of times that node i was visited where i denotes a child and p the parent to give an upper confidence bound for the average game value v_i. For puzzles, we added a third term, which represents a possible deviation of the child node (Chaslot *et al.*, 2006a; Coulom, 2007a). It contains the sum of the squared results so far $\left(\sum r^2\right)$ achieved in the child node corrected by the expected results $n_i \times v_i^2$. A high constant D is added to ensure that nodes, which have been rarely explored, are considered uncertain. Below we describe two differences between puzzles and two-player games, which may affect the selection strategy.

First, the essential difference between puzzles and two-player games is the *range of values*. In two-player games, the outcome of a game is usually denoted by *loss*, *draw*, or *win*, i.e., $\{-1, 0, 1\}$. The average score of a node always stays within $[-1, 1]$. In a puzzle, an arbitrary score can be achieved that is not by definition within a preset interval. For example, in SameGame there are positions, which result in a

value above 5,000 points. As a first solution to this issue we may set the constants C and D in such a way that they are feasible for a certain interval (e.g., [0, 6000] in SameGame). A second solution would be to scale the values back into the above mentioned interval $[-1, 1]$, given a maximum score (e.g., 6,000 for a SameGame position). When the exact maximum score is not known a theoretical upper bound can be used. For instance, in SameGame a theoretical upper bound is to assume that all blocks have the same color. A direct consequence of such an upper bound is that due to the high upper bound, the game scores are located near to zero. It means that the constants C and D have to be set with completely different values compared to two-player games. We have opted for the first solution in our program.

A second difference is that puzzles do not have any *uncertainty on the opponent's play*. It means that the line of play has to be optimized without the hindrance of an opponent (Chaslot, 2010). Due to this, not only the average score but the top score of a move can be used as well. Based on manual tuning, we add the top score using a weight W with a value of 0.02 to the average score.

Here we remark that we follow Coulom (2007a) in choosing a move according to the selection strategy only if n_p reaches a certain threshold T (we set T to 10). As long as the threshold is not exceeded, the simulation strategy is used. The latter is explained in the next subsection.

3.3.2 Play-Out Step

The play-out step begins when we enter a position that is not part of the tree yet. Moves are randomly selected until the game ends. This succeeding step is called the play-out. In order to improve the quality of the play-outs, the moves are chosen quasi-randomly based on heuristic knowledge (Bouzy, 2005; Gelly *et al.*, 2006; Chen and Zhang, 2008). For SameGame, several simulation strategies exist.

We have proposed two simulation strategies, called TabuRandom and TabuColorRandom (Schadd *et al.*, 2008c). Both strategies aim at creating large groups of one color. In SameGame, creating large groups of blocks is advantageous. TabuRandom chooses a random color at the start of a play-out. The idea is not to allow to play this color during the play-out unless there are no other moves possible. With this strategy large groups of the chosen color are formed automatically. The new aspect in the TabuColorRandom strategy with respect to the previous strategy is that the chosen color is the color most frequently occurring at the start of the play-out. This may increase the probability of having large groups during the play-out. We also use the ϵ-greedy policy to occasionally deviate from this strategy (Sutton and Barto, 1998). Before the simulation strategy is applied, with probability ϵ a random move is played. Based on manual tuning, we chose $\epsilon = 0.003$.

An alternative simulation strategy for SameGame is Monte-Carlo with Roulette-Wheel Selection (MC-RWS) (Takes and Kosters, 2009). This strategy not only tries to maximize one group of a certain color, but also tries to create bigger groups of other colors. Tak (2010) showed that MC-RWS does not improve the score in SP-MCTS because it is computationally more expensive than TabuColorRandom.

3.3.3 Expansion Step

The expansion strategy decides on which nodes are stored in memory. Coulom (2007a) proposed to expand one child per play-out. With his strategy, the expanded node corresponds to the first encountered position that was not present in the tree. This is also the strategy we used for SP-MCTS.

3.3.4 Backpropagation Step

During the backpropagation step, the result of the play-out at the leaf node is propagated backwards to the root. Several backpropagation strategies have been proposed in the literature (Chaslot *et al.*, 2006b; Coulom, 2007a). The best results that we have obtained for SP-MCTS was by using the plain average of the play-outs. Therefore, we update (1) the average score of a node. Additional to this, we also update (2) the sum of the squared results because of the third term in the selection strategy (see Formula 3.1), and (3) the top score achieved so far.

3.3.5 Final Move Selection

The four steps are iterated until the time runs out.[5] When this occurs, a final move selection is used to determine which move should be played. In two-player games (with an analogous run-out-of-time procedure) the best move according to this strategy is played by the player to move. The opponent has then time to calculate his response. But in puzzles this can be done differently. In puzzles it is not required to wait for an unknown reply of an opponent. It is therefore possible to perform one large search from the initial position and then play all moves at once. With this approach all moves at the start are under consideration until the time for SP-MCTS runs out. It has to be investigated whether this approach outperforms an approach that allocates search time for every move. These experiments are presented in Subsection 3.5.3.

3.3.6 Randomized Restarts

We observed that it is important to generate deep trees in SameGame (see Subsection 3.5.2). However, by exploiting the most-promising lines of play, the SP-MCTS can be caught in local maxima. So, we randomly restart SP-MCTS with a different seed to overcome this problem. Because no information is shared between the searches, they explore different parts of the search space. This method resembles root parallelization (Chaslot *et al.*, 2008b).

Root parallelization is an effective way of using multiple cores simultaneously (Chaslot *et al.*, 2008b). However, we argue that root parallelization may also be used for avoiding local maxima in a single-threaded environment. Because there is no actual parallelization, we call this *randomized restarts*. Subsection 3.5.3 shows that randomized restarts are able to increase the average score significantly.

[5] In general, there is no time limitation for puzzles. However, a time limit is necessary to make testing possible.

3.4 The Cross-Entropy Method

Choosing the correct SP-MCTS parameter values is important for its success. For instance, an important parameter is the C constant which is responsible for the balance between exploration and exploitation. Optimizing these parameters manually may be a hard and time-consuming task. Although it is possible to make educated guesses for some parameters, for other parameters it is not possible. Specially hidden dependencies between the parameters complicate the tuning process. Here, a learning method can be used to find the best values for these parameters (Sutton and Barto, 1998; Beal and Smith, 2000).

The *Cross-Entropy Method* (CEM) (Rubinstein, 2003) has successfully tuned parameters of an MCTS program in the past (Chaslot *et al.*, 2008c). CEM is an evolutionary optimization method, related to *Estimation-of-Distribution Algorithms* (EDAs) (Mühlenbein, 1997). CEM is a population-based learning algorithm, where members of the population are sampled from a parameterized probability distribution (e.g., Gaussian, Binomial, Bernoulli, etc.). This probability distribution represents the range of possible solutions.

CEM converges to a solution by iteratively changing the parameters of the probability distribution (e.g., μ and σ for a Gaussian distribution). An iteration consists of three main steps. First, a set S of vectors $x \in X$ is drawn from the probability distribution, where X is some parameter space. These parameter vectors are called samples. In the second step, each sample is evaluated and gets assigned a fitness value. A fixed number of samples within S having the highest fitness are called the *elite samples*. In the third step, the elite samples are used to update the parameters of the probability distribution.

Generally, CEM aims to find the optimal solution x^* for a learning task described in the following form

$$x^* \leftarrow \underset{x}{\operatorname{argmax}}\, f(x), \tag{3.2}$$

where x^* is a vector containing all parameters of the (approximately) optimal solution. f is a fitness function that determines the performance of a sample x (for SameGame this is the average number of points scored on a set of positions). The main difference of CEM to traditional methods is that CEM does not maintain a single candidate solution, but maintains a distribution of possible solutions.

There exist two methods for generating samples from the probability distribution, (1) random guessing and (2) distribution focusing (Rubinstein, 2003). *Random guessing* straightforwardly creates samples from the distribution and selects the best sample as an estimate for the optimum. If the probability distribution peaked close to the global optimum, random guessing may obtain a good estimate. If the distribution is rather uniform, the random guessing is unreliable. After drawing a moderate number of samples from a distribution, it may be impossible to give an acceptable approximation of x^*, but it may be possible to obtain a better sampling distribution. To modify the distribution to form a peak around the best samples is called *distribution focusing*. Distribution focusing is the central idea of CEM (Rubinstein, 2003).

Table 3.1: Effectiveness of the simulation strategies.

	Random	TabuRandom	TabuColorRandom
Average Score	2,069	2,737	3,038
StdDev	322	445	479

When starting CEM, an initial probability distribution is required. Chaslot *et al.* (2008c) used a Gaussian distribution and proposed that for each parameter, the mean μ of the corresponding distribution is equal to the average of the lower and upper bound of that parameter. The standard deviation σ is set to half the difference between the lower and upper bound (cf. Tak, 2010).

3.5 Experiments and Results

In this section we test SP-MCTS in SameGame. All experiments were performed on an AMD64 2.4 GHz computer. Subsection 3.5.1 shows quality tests of the two simulation strategies TabuRandom and TabuColorRandom. Thereafter, the results of manual parameter tuning are presented in Subsection 3.5.2. Subsequently, Subsection 3.5.3 gives the performance of the randomized restarts on a set of 250 positions. In Subsection 3.5.4, it is investigated whether it is beneficial to exhaust all available time at the first move. Next, in Subsection 3.5.5 the parameter tuning by CEM is shown. Finally, Subsection 3.5.6 compares SP-MCTS to the other approaches.

3.5.1 Simulation Strategy

In order to test the effectiveness of the two simulation strategies, we used a test set of 250 randomly generated positions.[6] We applied SP-MCTS without randomized restarts for each position until 10 million nodes were reached in memory. These runs typically take 5 to 6 minutes per position. The best score found during the search is the final score for the position. The constants C and D were set to 0.5 and 10,000, respectively. The results are shown in Table 3.1.

Table 3.1 shows that the TabuRandom strategy has a significantly better average score (i.e., 700 points) than plain random. Using the TabuColorRandom strategy the average score is increased by another 300 points. We observe that a low standard deviation is achieved for the random strategy. In this case, it implies that all positions score almost equally low. The proposed TabuColorRandom strategy has also been successfully applied in Nested Monte-Carlo Search (Cazenave, 2009) and HGSTS (Edelkamp *et al.*, 2010).

3.5.2 Manual Parameter Tuning

This subsection presents the parameter tuning in SP-MCTS. Three different settings were used for the pair of constants (C; D) of Formula 3.1, in order to investigate which balance between exploitation and exploration gives the best results. These

[6] The test set can be found at http://www.personeel.unimaas.nl/maarten-schadd/TestSet.txt

constants were tested with three different time controls on the test set of 250 positions, expressed by a maximum number of nodes. The short time control refers to a run with a maximum of 10^5 nodes in memory. At the medium time control, 10^6 nodes are allowed in memory, and for a long time control 5×10^6 nodes are allowed. We have chosen to use nodes in memory as measurement to keep the results hardware-independent. The parameter pair (0.1; 32) represents *exploitation*, (1; 20,000) performs *exploration*, and (0.5; 10,000) is a balanced setting.

Table 3.2 shows the performance of the SP-MCTS approach for the three time controls. The short time control corresponds to approximately 20 seconds per position. The best results are achieved by exploitation. The score is 2,552. With this setting the search is able to build trees that have on average the deepest leaf node at ply 63, implying that a substantial part of the chosen line of play is inside the SP-MCTS tree. Also, we observe that the other two settings are not generating a deep tree.

For the medium time control, the best results were achieved by using the balanced setting. It scores 2,858 points. Moreover, Table 3.2 shows that the average score of the balanced setting increases most compared to the short time control, viz. 470. The balanced setting is able to build substantially deeper trees than at the short time control (37 vs. 19). An interesting observation can be made by comparing the score of the exploration setting for the medium time control to the exploitation score in the short time control. Even with 10 times the amount of time, exploration is not able to achieve a significantly higher score than exploitation.

The results for the long experiment are that the balanced setting again achieves the highest score with 3,008 points. The deepest node in this setting is on average at ply 59. However, the exploitation setting only scores 200 points fewer than the balanced setting and 100 points fewer than exploration.

Table 3.2: Results of SP-MCTS for different settings.

10^5 nodes (∼20 seconds)	Exploitation (0.1; 32)	Balanced (0.5; 10,000)	Exploration (1; 20,000)
Average Score	2,552	2,388	2,197
Standard Deviation	572	501	450
Average Depth	25	7	3
Average Deepest Node	63	19	8
10^6 nodes (∼200 seconds)	(0.1; 32)	(0.5; 10,000)	(1; 20,000)
Average Score	2,674	2,858	2,579
Standard Deviation	607	560	492
Average Depth	36	14	6
Average Deepest Node	71	37	15
5×10^6 nodes (∼1,000 seconds)	(0.1; 32)	(0.5; 10,000)	(1; 20,000)
Average Score	2,806	3,008	2,901
Standard Deviation	576	524	518
Average Depth	40	18	9
Average Deepest Node	69	59	20

From the results presented we may draw two conclusions. First, it is important to have a deep search tree. Second, exploiting local maxima can be more advantageous than searching for the global maximum when the search only has a small amount of time.

3.5.3 Randomized Restarts

This subsection presents the performance tests of the randomized restarts on the set of 250 positions. We remark that the experiments are time constrained. Each experiment could only use 5×10^5 nodes in total and the restarts distributed these nodes uniformly among the number of searches. It means that a single search can take all 5×10^5 nodes, but that two searches can only use 2.5×10^5 nodes each. We used the exploitation setting (0.1; 32) for this experiment. The results are depicted in Figure 3.2.

Figure 3.2 indicates that already with two searches instead of one, a significant performance increase of 140 points is achieved. Furthermore, the maximum average score of the randomized restarts is at ten threads, which uses 5×10^4 nodes for each search. Here, the average score is 2,970 points. This result is almost as good as the best score found in Table 3.2, but with the difference that the randomized restarts together used one tenth of the number of nodes. After 10 restarts the performance decreases because the generated trees are not deep enough.

3.5.4 Time Control

This subsection investigates whether it is better to exhaust all available time at the initial position or to distribute the time uniformly for every move. Table 3.3 shows the average score on 250 random positions with five different time settings.

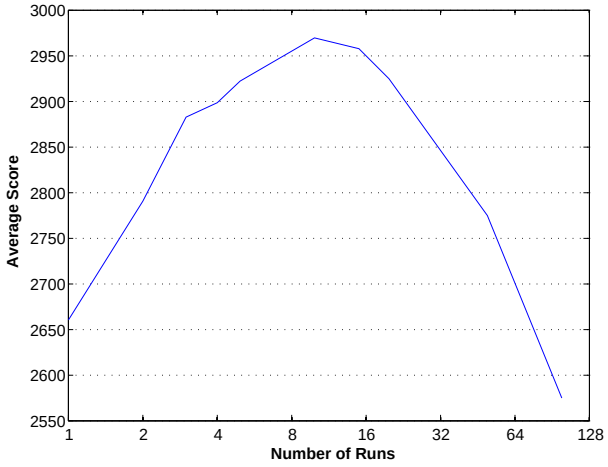


Figure 3.2: The average score for different settings of randomized restarts.

When SP-MCTS is applied for every move, this time is divided by the average game length (64.4). It means that depending on the number of moves, the total search time varies. These time settings are exact in the case that SP-MCTS is applied per game. This experiment was performed in collaboration with Tak (2010).

Table 3.3: Average score on 250 positions using different time control settings (Tak, 2010).

Time in seconds	∼5	∼10	∼20	∼30	∼60
SP-MCTS per game	2,223	2,342	2,493	2,555	2,750
SP-MCTS per move	2,588	2,644	2,742	2,822	2,880

Table 3.3 shows that distributing the time uniformly for every move is the better approach. For every time setting a higher score is achieved when searching per move. The difference in score is largest for 5 seconds, and smallest for 60 seconds. It is an open question whether for longer time settings it may be beneficial to exhaust all time at the initial position.

3.5.5 CEM Parameter Tuning

In the next series of experiments we tune SP-MCTS with CEM. The experiments have been performed in collaboration with Tak (2010). The following settings for CEM were used. The sample size is equal to 100, the number of elite samples is equal to 10. Each sample plays 30 games with 1 minute thinking time for each game. The 30 initial positions are randomly generated at the start of each iteration. The fitness of a sample is the average of the scores of these games. The five parameters tuned by CEM are presented in Table 3.4. C, D, T and W were described in Subsection 3.3.1. The ϵ parameter was described in Subsection 3.3.2. The CEM-tuned parameters differ significantly from the manually tuned ones. For more results on tuning the parameters, we refer to Tak (2010).

Table 3.4: Parameter tuning by CEM (Tak, 2010).

Parameter	Manual	CEM per game	CEM per move
C	0.1	5.96	4.31
D	32	67.98	96.67
T	10	13	11
W	0.02	0.49	0.28
ϵ	0.003	0.00007	0.00007[7]

To determine the performance of the parameters found by CEM an independent test set of 250 randomly created positions was used. Five different time settings were investigated. Table 3.5 shows the results of the CEM experiments. Here, the search time is distributed uniformly for every move.

[7] This parameter was not tuned again because it was obvious that the optimal weight is close to or equal to zero.

Table 3.5: Average scores of CEM tuning (Tak, 2010).

Time in seconds	∼5	∼10	∼20	∼30	∼60
Manual tuned	2,588	2,644	2,742	2,822	2,880
Average Depth	22.7	27.4	30.3	32.8	35.9
Average Deepest Node	31.8	36.8	39.1	41.4	44.3
CEM tuned	2,652	2,749	2,856	2,876	2,913
Average Depth	4.9	5.4	6.2	6.8	9.1
Average Deepest Node	9.0	10.2	12.2	13.5	19.2

Table 3.5 shows that for every time setting CEM is able to improve the score. This demonstrates the difficulty of finding parameters manually in a high-dimensional parameter space. The CEM-tuned parameters are more explorative than the manually tuned parameters. This difference may be due to the fact that the CEM parameters are tuned for the "per move" time control setting. The average depth and average deepest node achieved by the CEM parameters are closest to the results of the balanced setting in Table 3.2.

3.5.6 Comparison on the Standardized Test Set

Using two hours per position, we tested SP-MCTS on the standardized test set. We tested three different versions of SP-MCTS, subsequently called SP-MCTS(1), SP-MCTS(2), and SP-MCTS(3). SP-MCTS(1) builds one large tree at the start and uses the exploitation setting (0.1; 32) and randomized restarts, which applied 1,000 runs using 100,000 nodes for each search thread. SP-MCTS(2) uses the same parameters as SP-MCTS(1), but distributes its time per move. SP-MCTS(3) distributes its time per move and uses the parameters found by CEM. Table 3.6 compares SP-MCTS with other approaches, which were described in Subsection 3.1.3.

SP-MCTS(1) outperformed DBS on 11 of the 20 positions and was able to achieve a total score of 73,998. This was the highest score on the test set at the point of our publication (cf. Schadd *et al.*, 2008c). SP-MCTS(2) scored 76,352 points, 2,354 more than SP-MCTS(1). This shows that it is important to distribute search time for every move. SP-MCTS(3) achieved 78,012 points, the third strongest method at this point of time. All SP-MCTS versions are able to clear the board for all 20 positions.[8] This confirms that a deep search tree is important for SameGame as shown in Subsection 3.5.2.

The two highest scoring programs (1) SPURIOUS AI and (2) HGSTS achieved more points than SP-MCTS. We want to give the following remarks on these impressive scores. (1) SPURIOUS AI is memory intensive and it is unknown what time settings were used for achieving this score. (2) HGSTS utilized the graphics processing unit (GPU), was optimized for every position in the standardized test set, and applied our TabuColorRandom strategy. Moreover, the scores of HGTS were not independently verified to be correct.

[8] The best variations can be found at the following address:
http://www.personeel.unimaas.nl/maarten-schadd/SameGame/Solutions.html

Table 3.6: Comparing the scores on the standardized test set.

Position no.	DBS	**SP-MCTS(1)**	**SP-MCTS(2)**	MC-RWS
1	2,061	2,557	2,969	2,633
2	3,513	3,749	3,777	3,755
3	3,151	3,085	3,425	3,167
4	3,653	3,641	3,651	3,795
5	3,093	3,653	3,867	3,943
6	4,101	3,971	4,115	4,179
7	2,507	2,797	2,957	2,971
8	3,819	3,715	3,805	3,935
9	4,649	4,603	4,735	4,707
10	3,199	3,213	3,255	3,239
11	2,911	3,047	3,013	3,327
12	2,979	3,131	3,239	3,281
13	3,209	3,097	3,159	3,379
14	2,685	2,859	2,923	2,697
15	3,259	3,183	3,295	3,399
16	4,765	4,879	4,913	4,935
17	4,447	4,609	4,687	4,737
18	5,099	4,853	4,883	5,133
19	4,865	4,503	4,685	4,903
20	4,851	4,853	4,999	4,649
Total:	72,816	**73,998**	**76,352**	76,764

Position no.	Nested MC	**SP-MCTS(3)**	SPURIOUS AI	HGSTS
1	3,121	2,919	3,269	2,561
2	3,813	3,797	3,969	4,995
3	3,085	3,243	3,623	2,858
4	3,697	3,687	3,847	4,051
5	4,055	4,067	4,337	4,633
6	4,459	4,269	4,721	5,003
7	2,949	2,949	3,185	2,717
8	3,999	4,043	4,443	4,622
9	4,695	4,769	4,977	6,086
10	3,223	3,245	3,811	3,628
11	3,147	3,259	3,487	2,796
12	3,201	3,245	3,851	3,710
13	3,197	3,211	3,437	3,271
14	2,799	2,937	3,211	2,432
15	3,677	3,343	3,933	3,877
16	4,979	5,117	5,481	6,074
17	4,919	4,959	5,003	5,166
18	5,201	5,151	5,463	6,044
19	4,883	4,803	5,319	5,019
20	4,835	4,999	5,047	5,175
Total:	77,934	**78,012**	84,414	84,718

3.6 Chapter Conclusions and Future Research

In this chapter we proposed a new MCTS variant called Single-Player Monte-Carlo Tree Search (SP-MCTS). We adapted MCTS by two modifications resulting in SP-MCTS. The modifications are (1) in the selection strategy and (2) in the backpropagation strategy. Below we provide five observations and one conclusion.

First, we observed that our TabuColorRandom strategy significantly increased the score of SP-MCTS in SameGame. Compared to the pure random play-outs, an increase of 50% in the average score is achieved. The proposed TabuColorRandom strategy has also been successfully applied in Nested Monte-Carlo Search (Cazenave, 2009) and HGSTS (Edelkamp *et al.*, 2010). Second, we observed that exploiting works better than exploring at short time controls. At longer time controls a balanced setting achieves the highest score, and the exploration setting works better than the exploitation setting. However, exploiting the local maxima still leads to comparable high scores. Third, with respect to the randomized restarts, we observed that for SameGame combining a large number of small searches can be more beneficial than performing one large search. Fourth, it is better to distribute search time equally over the consecutive positions than to invest all search time at the initial position. Fifth, CEM is able to find better parameter values than manually tuned parameter values. The parameters found by CEM resemble a balanced setting. They were tuned for applying SP-MCTS for every move, causing that deep trees are less important.

The main conclusion is that we have shown that MCTS is applicable to a one-player deterministic perfect-information game. Our variant, SP-MCTS, is able to achieve good results in SameGame. Thus, SP-MCTS is a worthy alternative for puzzles where a good admissible estimator cannot be found.

There are two directions of future research for SP-MCTS. The first direction is to test several enhancements in SP-MCTS. We mention two of them. (1) The selection strategy can be enhanced with RAVE (Gelly and Silver, 2007) or progressive widening (Chaslot *et al.*, 2008d; Coulom, 2007a). (2) This chapter demonstrated that combining small searches can achieve better scores than one large search. However, there is no information shared between the searches. This can be achieved by using a transposition table, which is not cleared at the end of a small search. The second direction is to apply SP-MCTS to other domains. For instance, we could test SP-MCTS in puzzles such as Morpion Solitaire and Sudoku (Cazenave, 2009) and Single-Player General Game Playing (Méhat and Cazenave, 2010). Other classes of one-player games, with non-determinism or imperfect information, could be used as test domain for SP-MCTS as well.

Chapter 4

Proof-Number Search with Endgame Databases

This chapter is an updated and abridged version of the following publications:

1. Schadd, M.P.D., Winands, M.H.M., Bergsma, M.H.J., Uiterwijk, J.W. H.M., and Herik, H.J. van den. (2007a). Fanorona is a Draw. *ICGA Journal*, Vol. 30, No. 1, pp. 43–45.
2. Schadd, M.P.D., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bergsma, M.H.J. (2007b). Best Play in Fanorona Leads to Draw. *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)* (eds., P. Wang *et al.*), pp. 635–642, World Scientific Publishing Co. Pte. Ltd., Singapore.
3. Schadd, M.P.D., Winands, M.H.M., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Bergsma, M.H.J. (2008d). Best Play in Fanorona Leads to Draw. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 369–387.

This chapter answers the second research question by weakly solving the two-player deterministic perfect-information board game Fanorona. We propose to use a search-based approach that establishes the game-theoretic value of Fanorona. The search-based approach is a combination of a mate-solver (Proof-Number (PN) search) and pre-constructed endgame databases. We examine the tradeoff between time spent on backward search (i.e., creating the endgame databases) and time invested in forward search (i.e., mate-solvers using the created endgame databases). Based on the analysis of the game, solving Fanorona and its variants is possible by a well-chosen combination of PN search and endgame databases. Moreover, the behavior of the PN search is investigated when solving different variants of the game.

The chapter is organized as follows. Section 4.1 discusses the notion of solving games. Next, Section 4.2 provides background information and the rules of the game of Fanorona. Subsequently, Section 4.3 presents various characteristics of the game. Section 4.4 discusses the application of a retrograde analysis in Fanorona.

The proof-number search variant, PN^2, is explained in Section 4.5. Then, Section 4.6 presents the results of a well-tuned combination of PN^2 search and the corresponding databases. Section 4.7 describes the verification of the results. Finally, Section 4.8 provides the conclusions and gives future research topics.

4.1 Solving Games

Already for half a century, building strong game-playing programs is one of the main targets of AI researchers. The principal aim is to witness the "intelligence" of computers. A second aim is to establish the *game-theoretic value* of a game, i.e., the outcome of the game when all participants play optimally (Van den Herik, Uiterwijk, and Van Rijswijk, 2002). The game-theoretic value indicates whether a game is won, lost, or drawn from the perspective of the player who has to move first.

Pursuing the second aim is an exciting task; there, game solvers are looking for new techniques and new achievements in research. For solving a game, three "layered" definitions exist (cf. Allis, 1994).

- **Ultra-weakly solved**: For the initial position, the game-theoretic value has been determined.
- **Weakly solved**: For the initial position, a strategy has been determined to obtain at least the game-theoretic value of the game.
- **Strongly solved**: For all legal positions, a strategy has been determined to obtain at least the game-theoretic value of the position.

In the last 20 years, quite a number of games have been solved (cf. Van den Herik *et al.*, 2002). Below, we provide examples for each definition. (1) The game of Hex is an instance of an ultra-weakly solved game (Anshelevich, 2000). The proof was published by Nash (1952). (2) The following games were weakly solved: Qubic (Patashnik, 1980; Allis and Schoo, 1992), Go-Moku (Allis, 1994; Allis, Van den Herik, and Huntjes, 1996), Nine Men's Morris (Gasser, 1995), Domineering (Breuker, Uiterwijk, and Van den Herik, 2000; Bullock, 2002), and Renju (Wágner and Virág, 2001). Checkers is the latest addition to this list (Schaeffer *et al.*, 2007; Schaeffer, 2007). (3) Recently, we saw considerable development in "layer" three: Kalah (Irving, Donkers, and Uiterwijk, 2000), Awari (Romein and Bal, 2003) and, Connect-Four (Tromp, 2008) were strongly solved.

Different methods can be used for solving games, which we divide into three classes: knowledge methods, special search methods and endgame databases.[1]

(1) Knowledge methods are used to prune the search tree. Allis (1988) applied knowledge rules in Connect-Four that made it possible to determine the game-theoretic value for non-terminal positions. Using these rules the game of Connect-Four was solved.

(2) Special search methods (mate-solvers) are used to search the space efficiently. For instance, Threat-Space Search (Allis, Van den Herik, and Huntjes, 1993) was

[1] It is possible to define different classes. Heule and Rothkrantz (2007) use five classes to distinguish solving methods.

introduced in Go-Moku. This technique searches as if the opponent is allowed to play all counter moves to a threat at once. Using this technique, a series of threat sequences to win the game can be established in such that it is indifferent what the opponent plays. The technique reduces the size of the search tree in such a way that it was possible to solve the game (Allis *et al.*, 1996). Other examples of mate-solvers are Proof-Number Search (Allis *et al.*, 1994) and λ-search (Thomson, 2000).

(3) Endgame databases are used to store the game-theoretic value of each possible endgame position. Retrograde analysis (Ströhlein, 1970) is a method to solve endgame positions iteratively, starting with the terminal ones. For instance, Romein and Bal (2003) were able to solve every possible Awari position using parallel retrograde analysis, thus solving the game strongly.

4.2 Fanorona

Fanorona is a board game with its roots in Madagascar (Montgomery, 1886). It is derived from the game "Alquerque", which might be over 3,000 years old. Below, we explain the rules of Fanorona. The explanation is based on the rules given in Bell (1980), and in Chauvicourt and Chauvicourt (1980). There exist some variations on the rules, but we will focus on the most common variant. The goal of the game is to capture all opponent pieces. The game is a draw if neither player succeeds. Fanorona has three standard versions: *Fanoron-Telo*, *Fanoron-Dimyand*, and *Fanoron-Tsivy*. The difference between these versions is the board size. Fanoron-Telo is played on a 3×3 board and the difficulty of this game can be compared to that of Tic-Tac-Toe. Fanoron-Dimyand is played on a 5×5 board and Fanoron-Tsivy is played on a 5×9 board. We call Fanoron-Tsivy from now on *Fanorona*, because it is the best-known board size and the main subject of this chapter.

This section is organized as follows. Subsection 4.2.1 introduces the board on which Fanorona is played. Next, Subsection 4.2.2 explains the movement rules. Finally, Subsection 4.2.3 describes the ending conditions of Fanorona.

4.2.1 Board

The Fanorona board consists of lines and intersections. A line represents the path that a piece can move along during the game. There are weak and strong intersections. On a weak intersection, it is only possible to move a piece horizontally and vertically, while on a strong intersection, it is also possible to move a piece diagonally. Figure 4.1 shows a weak intersection at **e2** and a strong intersection at **e3**. A piece can only move from one intersection to an adjacent intersection. In the initial position, each player has 22 pieces. They are placed as shown in Figure 4.1. Players move alternately; White plays first. Figure 4.2 shows two non-standard board sizes, the 5×7 and the 7×5 board. We remark that a game played on the 7×5 board is totally different from a game played on a 5×7 board. Rotating the 7×5 board to a 5×7 board creates a Fanorona game with a different 5×7 initial position (e.g., left Black, right White). It is standard in Fanorona that the white pieces are below the black pieces.

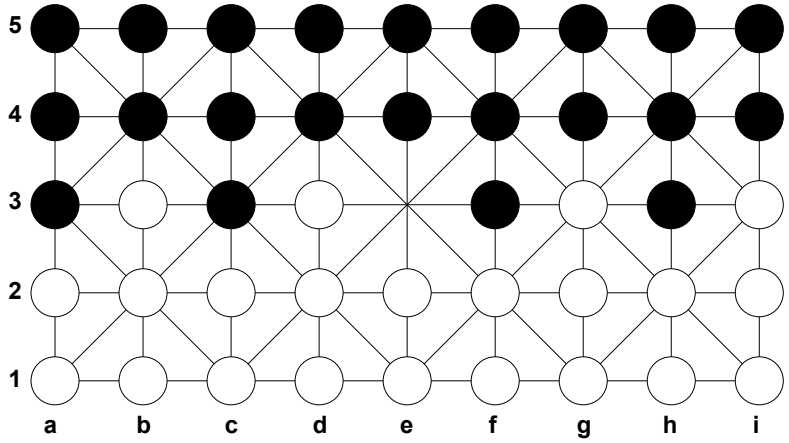
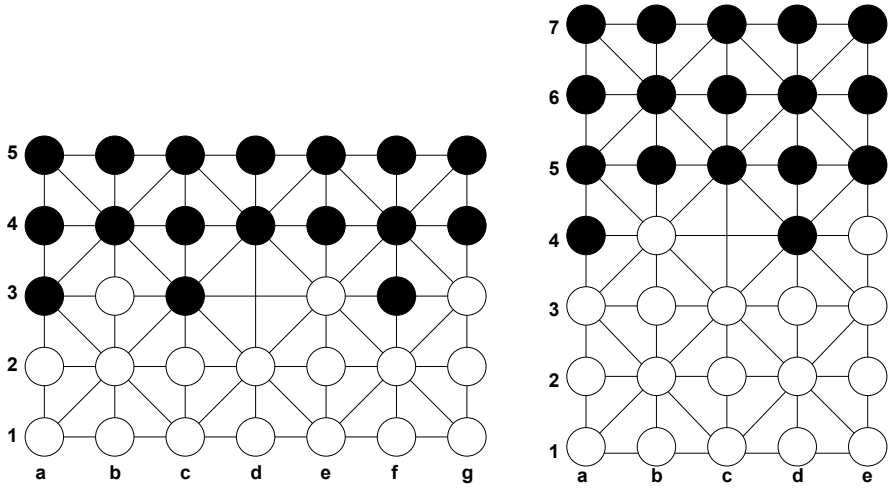


Figure 4.1: The initial position of a Fanorona game.



(a) 5x7: Initial position (b) 7x5: Initial position

Figure 4.2: Initial positions for the 5×7 and 7×5 board.

4.2.2 Movement

We distinguish two kinds of moves, capturing and non-capturing moves. Capturing moves are *obliged* and have to be played, if possible, above non-capturing (*paika*) moves. We start explaining capturing moves because thereafter describing paika moves is straightforward.

Capturing implies removing one or more pieces of the opponent. It can be done in two different ways, either (1) by approach or (2) by withdrawal. An *approach* is the movement of the capturing piece to an intersection adjacent to an opponent piece

provided that the opponent piece is situated on the continuation of the capturing piece's movement line. A *withdrawal* works analogously as an approach but the difference is that the movement is away from the opponent piece. When an opponent piece is captured, all opponent pieces in line with that piece and directly behind that piece (i.e., there is no interruption by an empty intersection or an own piece) are captured as well.

Figure 4.3 shows how the capturing mechanism works. We start with a straightforward capturing move. In the given position, White can capture Black's piece on **d2** by moving the white piece from **b2** to **c2**. By this move, Black's piece on **e2** is captured as well. **g2** is not captured because there is no black piece on **f2**. This is called *capturing by approach* because the white piece is moved towards the black piece on **d2**.

White can also *capture by withdrawal* if the white piece is moved from **f4** to **e4**. This is allowed because the white piece is moving away from the black piece on **g4**. The piece on **i4** is not captured because there is a white piece on **h4** interrupting the line.

White cannot capture **c4** with **f4** because for a capturing move the own piece has to be next to the one captured after movement: **f4** is too far away to capture **c4**. In order to allow capturing, the piece has to be moved to an intersection adjacent to the captured piece, if it is approached.

Also, White cannot capture **c4** with **b2** (moving diagonally) because **c4** is not on the extension of a movement line from **b2**. Only pieces can be captured that are located in the extension of the movement line of the capturing piece. Thus, capturing "around a corner" is not allowed.

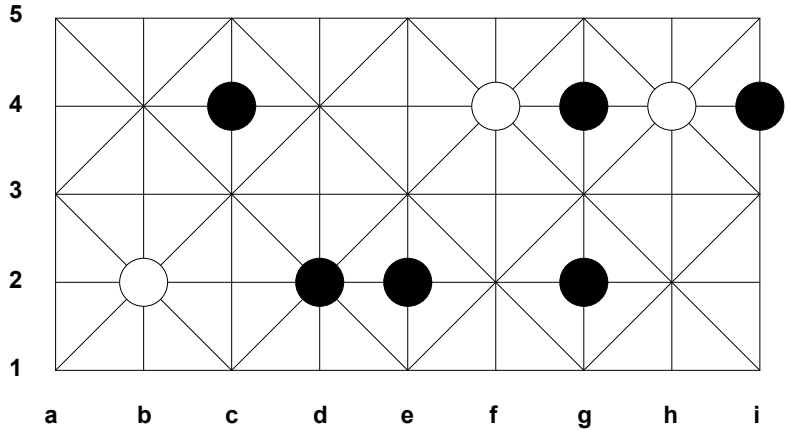


Figure 4.3: An example position during a Fanorona game.

To denote a capturing by approach, we define the following notation: **b2-c2A** meaning that the piece on **b2** moves to **c2** and approaches (A) the piece on **d2**. For a withdrawal, the letter "W" is used. If a piece is moved without capturing any opponent piece, no letter "A" or "W" is used.

As in checkers, it is allowed to continue capturing with the same piece as long as possible. We call this extended capturing. Figure 4.3 shows that White can capture **c4** with the move: **b2-c2A-c3A**. (Even if a move consists of multiple movements of one single piece it still counts as a single move.) Although a player is obliged to prefer a capturing move above a non-capturing move, the player may stop capturing after any number of opponent pieces are captured. This rule is different from the checkers rule where stopping a capturing sequence is not permitted. For instance, in Figure 4.3, White is allowed to stop early and play only **b2-c2A**.

There are three more rules concerning capturing pieces. (1) It is not allowed to capture by approach and withdrawal at the same time. This is the case at the initial position shown in Figure 4.1 where White could play **d3-e3** as an approach or a withdrawal. In such a situation, the player has to choose whether the current move is an approach or a withdrawal. (2) It is not allowed to make a capturing move in the same direction as the capturing move directly before. We illustrate this rule by referring to Figure 4.3. White is not allowed to play: **f4-e4W-d4A** because the white piece would move twice in a row in the same direction. A player is allowed to play a capturing movement in a previously chosen direction if a capturing movement in another direction is done in between. The last movement direction of the capturing move in the previous turn (i.e., before the last opponent move) does not influence possible capturing directions in the current turn. (3) The current capturing piece is not allowed to arrive on the same intersection twice during a capturing sequence. In Figure 4.3, White is not allowed to play **f4-e4W-e3A-f4W** because of this rule.

If no capturing move can be played, a non-capturing move is allowed. This is called a paika move and consists of moving one piece along a line to an adjacent intersection. White is not allowed to play the paika move **b2-b1** in the position depicted in Figure 4.3 because capturing is possible.

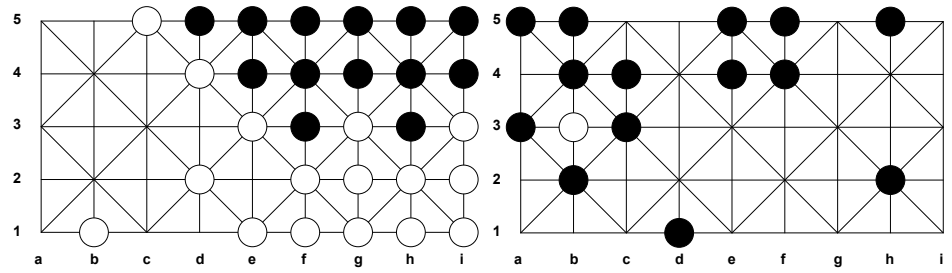
4.2.3 End of the Game

The player who first captures all opponent pieces wins the game. The game is a draw if no player is able to win. In practice this is achieved by repetition of the same position with the same player to move (Schadd, 2006).

There does not exist any documentation stating what the outcome of the game would be if a player is not able to move. If this occurs during a game, we define that a player who is not able to move forfeits the game. However, this situation rarely occurs and it is unlikely that the game-theoretic value would change if another outcome would be defined in this situation. This situation was encountered a few times when we solved Fanorona and its smaller variants. Two positions that were encountered during the search are depicted in Figure 4.4.

4.3 Analyzing Fanorona

Two important indicators for establishing the questions (a) whether games can be solved and (b) which methods may be successful are (1) the game-tree complexity and (2) the state-space complexity.



(a) No moves for Black (b) No moves for White

Figure 4.4: Two positions without legal moves for one side encountered during the search.

An approximation for the game-tree complexity can be computed by performing a self-play experiment. 10,000 games were played by two $\alpha\beta$ players, which performed a 4-ply deep search with a greedy evaluation function. The evaluation function consists of material difference and a random factor. By using these players, the average branching factor may be approximated accurately. It is still possible however, that the approximated average game length deviates significantly off the real average game length. On the one hand, in games where humans know early that the result is a draw, the $\alpha\beta$ players may shuffle pieces around until a repetition occurs. Only after the first repetition of a position, the game is counted as a draw. On the other hand, the $\alpha\beta$ players may make mistakes which leads to faster capturing of pieces. Therefore, the average game length is somewhat unreliable. We determined that the average game length of Fanorona is 43.81 moves (i.e., plies) and the average branching factor is 11.19 moves. This gives us a game-tree complexity of approximately 10^{46}. The state-space complexity has been computed as 10^{21} by using Formula 4.1 (explained in the next section). These numbers are comparable to those of checkers, which has a game-tree complexity and state-space complexity of 10^{31} and 10^{20}, respectively (Allis, 1994; Schaeffer *et al.*, 2007).

A typical game of Fanorona can be divided into two parts. In the first part of the game, mostly capturing moves are played until most pieces are captured. In the second part, the endgame, mainly paika moves are played. Figure 4.5 shows the ratio of capturing moves and paika moves as a function of the number of pieces on the board. This figure is based on 10,000 games played by $\alpha\beta$ players. In the initial position, 44 pieces are on the board.

Figure 4.5 shows that with 19 or more pieces on the board, a capturing move is to be expected. We define the endgame as the part of the game where more paika than capturing moves are played. Figure 4.5 indicates that the endgame starts when there are fewer than 13 pieces on the board.

Figure 4.6 shows the average branching factor as a function of the number of pieces on the board, based on the same 10,000 games. We see in the figure that when the game enters the endgame the branching factor increases again. The reason for the increase of the average number of possible moves in the endgame is the occurrence of paika moves. The combination of a long endgame and a local maximum of the branching factor results in an immense solution tree.

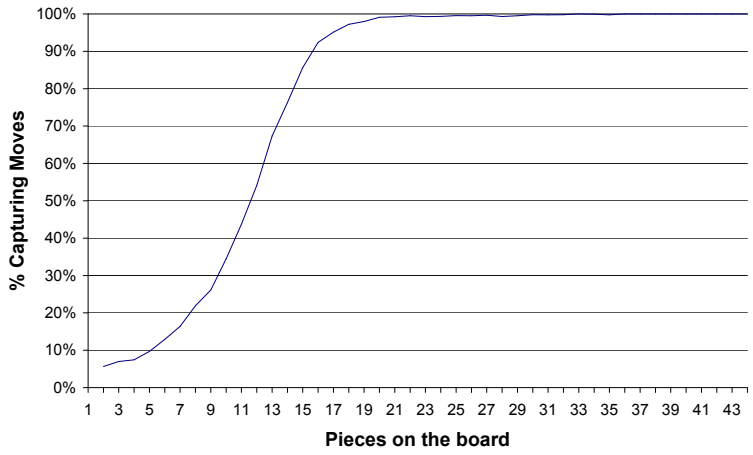


Figure 4.5: Ratio of capturing moves and paika moves as a function of the number of pieces on the board.

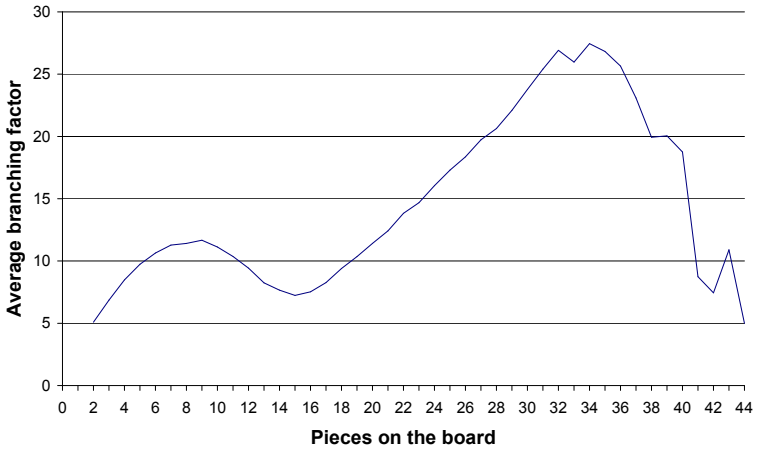


Figure 4.6: The average branching factor as a function of the number of pieces on the board.

Figure 4.7 shows the average number of pieces on the board as a function of the move number. Here we see that the number of pieces decreases fast at the start of the game. After 15 or more moves, the number of pieces only decreases slowly, implying that the endgame has started.

In order to cope well with the difference between both parts, different methods have been selected. For the endgame part, retrograde analysis has been selected to create endgame databases (Ströhlein, 1970; Van den Herik and Herschberg, 1985). There are three reasons, why endgame databases are a valuable tool for solving Fanorona. (1) Fanorona is a converging game (Van den Herik *et al.*, 2002). (2) A large part of the game consists of positions with only a few pieces on the board.

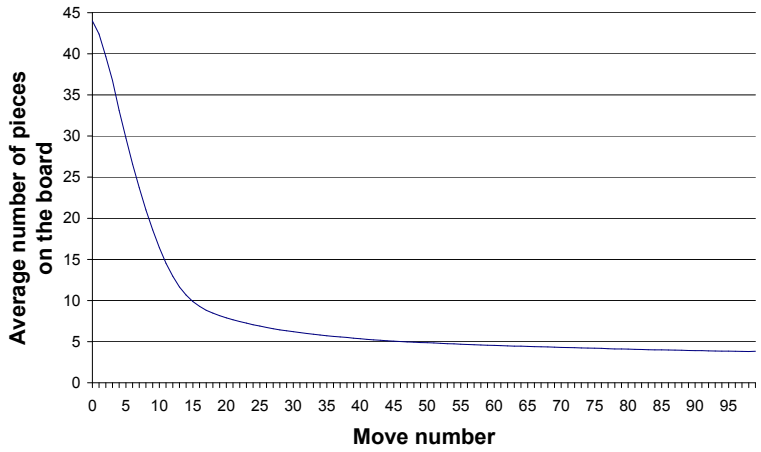


Figure 4.7: The average number of pieces as a function of the move number.

Table 4.1: Estimated complexities with increasing database size.

Database size	0	2	3	4	5
Average Game Length	43.81	42.60	40.23	36.23	31.62
Average Branching Factor	11.19	11.37	11.63	11.98	12.31
Log_{10} Game-Tree Complexity	45.96	44.97	42.88	39.07	34.48
Database size	6	7	8	9	10
Average Game Length	26.99	22.98	19.74	17.27	15.44
Average Branching Factor	12.59	12.82	13.02	13.26	13.53
Log_{10} Game-Tree Complexity	29.70	25.45	22.01	19.39	17.47

Fanorona has an average game length of 44 plies. Already after 21 plies there are on average fewer than 8 pieces on the board. (3) The endgame is not trivial. The branching factor has a local maximum in the endgame (see Figure 4.6) and there the game is converging more slowly.

Because of the long endgame where mostly paika moves are played, one would expect that an endgame database would decrease the size of the solution tree substantially. The expectation can be tested by doing a simulation. This was done in the following way.

A *virtual database* was used to finish a game early. For instance, if we assume that a 2-piece database is available, the game is stopped when a position with two pieces is reached. This is a terminal node because the outcome of the game can be retrieved from the virtual database. Using this approach, we can estimate the change of both the average game length and the branching factor as a direct consequence of using larger databases. Table 4.1 shows that the virtual database was able to reduce the game-tree complexity substantially. In this experiment, we see that the game-tree complexity decreases by a factor of more than 100 on average when the database size increases by one piece.

Because of the large number of positions, it is not possible to create an endgame database up to the initial position. Therefore, for the first part of the game, a dedicated search method is required. PN search (Allis *et al.*, 1994) has been chosen because the method is efficient for non-uniform trees. A non-uniform tree can be the result of many forced moves (e.g., capturing in Fanorona). Moreover, the use of an endgame database in the search tree makes the search tree non-uniform.

During the search the most-promising lines in the tree (i.e., lines where relatively the weakest resistance is expected) are examined first because PN search uses a best-first tree traversal to find the solution tree. The combination of endgame databases and the fact that Fanorona converges fast may make PN search an efficient technique for this game. In the next two sections, we describe the two methods, retrograde analysis and PN search, in detail.

4.4 Retrograde Analysis

Retrograde analysis is a method to create omniscient endgame databases for games (Ströhlein, 1970; Van den Herik and Herschberg, 1985). Such endgame databases proved to be vital to the strength of computer programs in quite a number of games (Schaeffer, 1997; Heinz, 1999). For instance, in the game of checkers endgame databases have been built for up to 10 pieces remaining on the board (Schaeffer *et al.*, 2003). The more board positions are stored in the endgame database, the earlier the search process can be terminated. Moreover, the method makes a deeper search possible in the middle game. Besides improving the playing strength of a program in the middle game, endgame databases can be used to solve a game as well. For instance, Romein and Bal (2003) solved the game of Awari by storing the complete state space in a database.

A requirement for retrograde analysis is an index function. This function has to be a one-to-one mapping. Making such a function efficient can save a significant amount of space in the database (Dekker, Van den Herik, and Herschberg, 1990; Lake, Schaeffer, and Lu, 1994). The function we used consists of two parts: (1) a function to transform the pieces on the board to a number, independent of the color of a piece and (2) a function to convert the order of black and white pieces to a number. The combined index function identifies uniquely all possible board positions disregarding symmetry. Such an index function is called gap-less. A gap-less function uses each index between the minimum and the maximum index. It is also invertible so that given an index, the corresponding board position can be computed.

Formula 4.1 shows the index function used. We denote the total number of pieces on the board by M and the position of a piece i by S_i (which is a number between 0 and 44). We define W as the total number of white pieces and W_i is the place of white piece i in the sequence of white and black pieces on the board (which is a number between 0 and $M-1$).

$$index = \sum_{i=1}^{M} \binom{S_i}{i} + \binom{45}{M} \times \sum_{i=1}^{W} \binom{W_i}{i} \tag{4.1}$$

Table 4.2: Database sizes up to 15 pieces.

# of Pieces	# of Positions
2	1,980
3	85,140
4	2,085,930
5	36,652,770
6	504,993,720
7	5,717,832,120
8	54,750,511,530
9	451,943,198,850
10	3,260,371,406,292
11	20,768,119,231,860
12	117,743,529,024,030
13	597,920,852,078,550
14	2,733,686,209,314,720
15	11,299,926,066,685,300

Table 4.2 shows the size of each database if the above index function is used.[2] Moreover, each position uses 2 bits of space on the hard disk, indicating that the position is a win, draw, or loss. So, for instance, we may state that the computation of the 9-piece database would be feasible on a regular desktop machine, where a total of 119.3 GB of hard disk space would be required.

For speeding up the creation process, paging was implemented. Paging is a technique that stores frequently used parts of the database in memory without writing them to the hard disk (Lake *et al.*, 1994). When data from a page is requested, the information can be retrieved from memory and no hard-disk access is required.

During this research all databases with 7 or fewer pieces were computed for Fanorona. The computation took two to three months and was performed on a regular desktop PC with a Pentium IV 3.0 GHz processor and 256 MB RAM of memory.

Table 4.3 shows the number of wins, draws, and losses in the databases for the 5×9 variant. Symmetric positions have been removed. *a*-*b* denotes that the player to move has *a* pieces and its opponent *b* pieces. Table 4.3 indicates that the player to move has an advantage. One might suspect that a player who is to move and has more pieces than the opponent would win the game. The results are that (1) in the 4-1 database the player to move cannot lose any position, (2) in the 5-1 and 6-1 databases the player to move can win every position. So, an advantage of more than three pieces is required for a forced win.

We have created databases for the smaller Fanorona boards as well. We have computed (1) all endgame databases up to 7 pieces for the 5×7 board, (2) all endgame databases up to 9 pieces for the 5×5 board, and (3) all endgame databases up to 8 pieces for the 3×9 board.

[2]Please note that $\binom{x}{y} = 0$ if $y > x$.

Table 4.3: Number of won, drawn, and lost positions in the databases for the 5×9 board. *a-b* indicates that the player with *a* stones is to move.

Db.	1-1	2-1	1-2	3-1	2-2
Win	158	10,366	717	149,458	127,756
Draw	334	398	3,231	91	79,012
Loss	26	6	6,822	1	17,386
Db.	1-3	4-1	3-2	2-3	1-4
Win	4,188	1,529,142	2,711,327	774,043	19,814
Draw	15,875	12	327,836	1,252,162	88,187
Loss	129,487	0	18,297	1,031,255	1,421,153
Db.	5-1	4-2	3-3	2-4	1-5
Win	12,223,788	30,095,407	24,137,779	4,180,200	81,728
Draw	0	426,350	13,955,354	7,926,733	391,405
Loss	0	32,491	2,644,731	18,447,315	11,750,655
Db.	6-1	5-2	4-3	3-4	2-5
Win	79,431,164	237,393,018	344,370,238	145,408,435	18,659,090
Draw	0	774,868	46,020,564	170,633,688	41,896,491
Loss	0	108,614	6,724,158	81,072,837	177,720,919
Db.	1-6				
Win	302,021				
Draw	1,509,775				
Loss	77,619,368				

4.5 Proof-Number Search

This section discusses the search procedure applied for weakly solving the game. The method used is Proof-Number (PN) search; it is briefly described in Subsection 4.5.1. A variant of PN search, called PN^2, is explained in Subsection 4.5.2.

4.5.1 PN Search

PN search is a best-first search algorithm especially suited for finding the game-theoretic value of a game (Allis *et al.*, 1994). Its aim is to prove the true value of the root of a tree. A tree can have three values: *true*, *false*, or *unknown*. In the case of a forced win, the tree is *proven* and its value is *true*. In the case of a forced loss or draw, the tree is *disproven* and its value is *false*. Otherwise, the value of the tree is *unknown*. In contrast to other best-first algorithms, PN search does not require a domain-dependent heuristic evaluation function to determine the most-promising node to be expanded next. In PN search, this node is usually called the *most-proving* node. PN search selects the most-proving node using two criteria: (1) the shape of the search tree (the branching factor of every internal node) and (2) the values of the leaves. These two criteria enable PN search to treat game trees with a non-uniform branching factor efficiently (Allis *et al.*, 1994). PN search exploits an irregular search tree by using a "least-work-first" strategy. An irregular

tree can be caused by capturing or blocking the opponent. Endgame databases can also influence the shape of the tree. For a number of application domains with a highly irregular tree structure, such as chess mating problems or the games of Qubic and Fanorona, PN search is more efficient than $\alpha\beta$-like algorithms (Plaat, 1996).

4.5.2 PN^2 Search

A disadvantage of PN search is that the whole search tree has to be stored in memory. Therefore, we use PN^2 as an algorithm to reduce memory requirements in PN search (Allis, 1994; Breuker, Uiterwijk, and Van den Herik, 2001b). PN^2 consists of two levels of PN search. The first level consists of a PN search (pn_1), which calls a PN search at the second level (pn_2) for an evaluation of the most-proving node of the pn$_1$-search tree. This pn$_2$ search is bound by a maximum number of nodes N to be stored in memory. In our implementation (analogously to Allis, 1994), N is equal to the size of the pn_1 tree. The pn$_2$ search is stopped when the number of nodes stored in memory exceeds N or the subtree is (dis)proven. After completion of the pn$_2$ search, the children of the root of the pn$_2$-search tree are preserved, but subtrees are removed from memory.[3]

The maximum size of the subtree for determining the PN numbers is an important factor for PN^2 search. With a larger tree, the search would be more directed but would take more time to compute. Allis (1994) proposed a variable size for the pn_2 tree that is dependent on the size of the pn_1 tree. Breuker *et al.* (2001b) used a fraction function to determine the size of the pn_2 tree. We did not use the latter approach because the fraction function is not efficient for large problems. We set the maximum size of the pn_2 tree equal to the size of the pn_1 tree.

4.6 Experiments and Results

This section presents the results obtained during our research. In Subsection 4.6.1, the game-theoretic values of Fanorona and its smaller board variants are given and the optimal solution for the 3×3 board is presented. Subsection 4.6.2 investigates the tradeoff between backward search and forward search. Finally, Subsection 4.6.3 discusses an interesting observation regarding the behavior of the proof and disproof numbers.

4.6.1 Solving Fanorona and its Smaller Variants

PN^2 search is used in combination with endgame databases to compute the game-theoretic value of Fanorona and its smaller variants. After some tuning, we arrived at the "ideal" combination for this configuration (i.e., 5×9 board on current hardware) viz. PN^2 with all databases including 7 or fewer pieces. By doing so, we were able to prove that Fanorona is a draw. For proving this, a total of 130,820,097,938 nodes were created during the search. Solving the initial position of Fanorona took more than a week when using our search-based approach on a computer with an AMD64

[3]This only holds if the root is not proven or disproven.

Table 4.4: The game-theoretic values for different board sizes.

Board Size	*Winner*	*DB Size (Pieces)*	*Nodes*
3×3	White	0	122
3×5	White	0	2,490
5×3	White	0	1,491
3×7	White	0	87,210
7×3	White	0	172,101
5×5	Draw	9	119,354
3×9	White	8	105,343
9×3	White	8	3,999,481
5×7	Black	7	72,826,963
7×5	White	7	1,107,756
5×9	Draw	7	130,820,097,938

2.6 GHz processor and 32 GB RAM of memory. The search proved that both moves **f2-e3A** and **d3-e3A** lead to a draw. Preliminary results suggest that by optimal play the moves **e2-e3A**, **d3-e3W** and **d2-e3A** will lead to a win for Black.

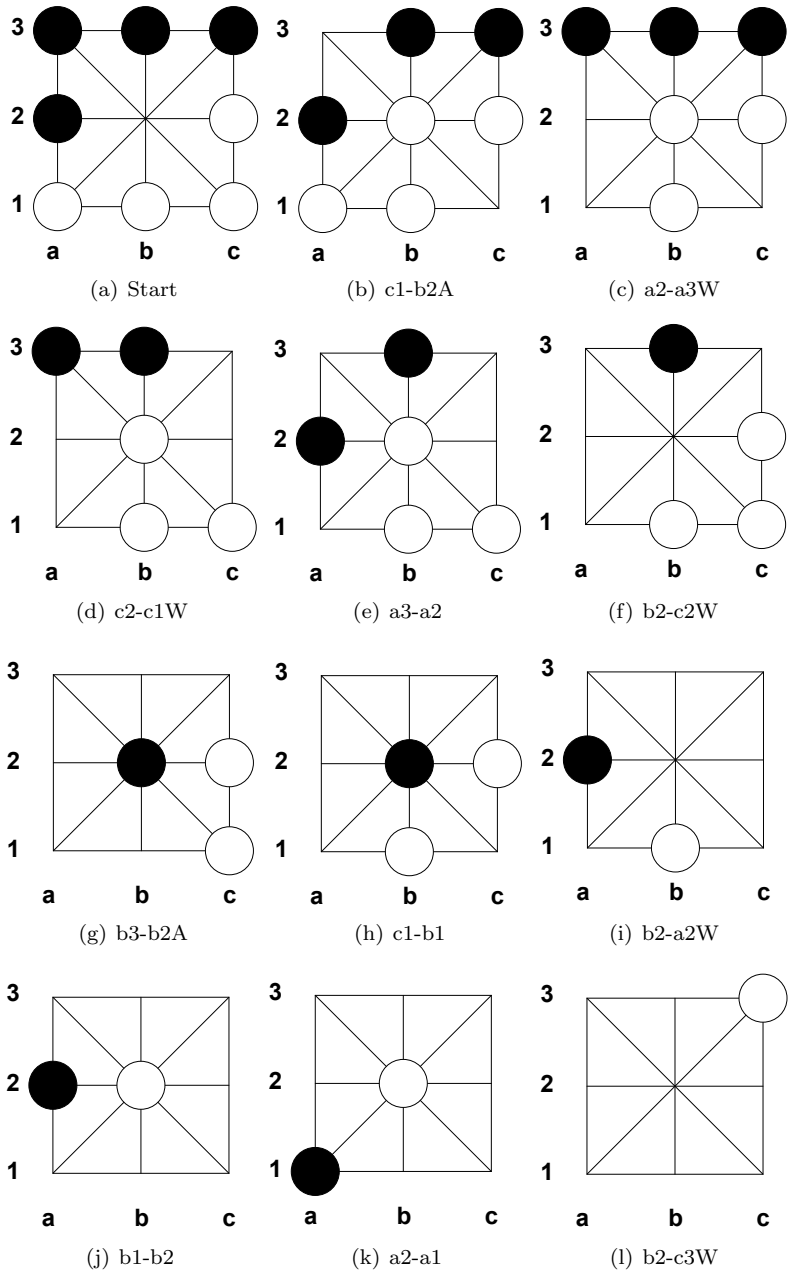
Furthermore, we have solved the smaller variants of the game as well. An overview of the results of Fanorona and all smaller boards is given in Table 4.4. The column labeled *DB size* indicates the maximum number of pieces for the databases used. The column labeled *Nodes* indicates the total number of created nodes.

For smaller boards we remark the following. If we have a look at Table 4.4, we see that all boards with a side equal to size 3 are a win for White. Thus, the starting player can exploit a narrow board and force a win. However, for most boards, with sides of at least size 5, White does not have this advantage anymore.

Table 4.4 shows differences between horizontal and vertical boards. The difference in number of nodes between the 5×7 and 7×5 boards (see Table 4.4 and Figure 4.2) can be explained by the fact that 5×7 is a win for Black (the second player), which is harder to prove. Furthermore, a substantial difference between the 3×9 and 9×3 boards can be observed. We conjecture that the average distance between the players' pieces is larger on vertical boards than on horizontal boards when entering the endgame. This would result in a slower convergence.

In order to provide insight into the strategies behind optimal play we show the solution for the 3×3 board. Figure 4.8 shows the game that is played if both players play optimally.

Below, some interesting positions in this game are discussed. In Subfigure 4.8(b), Black has no choice of movement because there is only one possible capturing move. In Subfigure 4.8(d), Black has two choices. The move **b3-c3** would lead to a loss in 1 because White would answer with **b2-a1W-a2A**. Subfigure 4.8(f) is the most interesting one. Black decides to stop the capturing sequence after the capturing of only one of the two possible pieces. The reason for this is to postpone the eventual loss by one move.



(a) Start (b) c1-b2A (c) a2-a3W

(d) c2-c1W (e) a3-a2 (f) b2-c2W

(g) b3-b2A (h) c1-b1 (i) b2-a2W

(j) b1-b2 (k) a2-a1 (l) b2-c3W

Figure 4.8: 3×3 board: White can force a win.

4.6.2 Tradeoff between Backward and Forward Search

There exists a tradeoff between time spent on backward search (i.e., creating the endgame databases) and time invested in forward searching (i.e., PN Search using the endgame databases to cut off the search). To demonstrate this, we take the 5×5 variant as example. Table 4.5 shows the time and the number of nodes required to compute different 5×5 databases. The number of positions in the databases and the time to compute the database increases exponentially. This table also shows the amount of time required to solve the 5×5 variant using different endgame databases. Here we see that the effect of an additional database diminishes fast.

Table 4.6 shows the effectiveness of endgame databases for the 7×5 variant. Because the 7×5 board is larger than the 5×5 board, only the databases up to 7 pieces were computed. The effectiveness of endgame databases for the 3×9 variant is presented in Table 4.7. For this variant, all databases with up to 8 pieces were computed. A remarkable phenomenon for this variant is that it is faster to solve with the 3-piece endgame database than with the 4-piece database. The following explanation can be given for this phenomenon. PN search develops the tree into the direction where the weakest resistance is expected. Because the endgame databases has a large influence on the branching factor, it may happen that the endgame database misdirects the PN search. The reason that this variant is solved faster with the 7-piece database than with the 8-piece database is that the problem size has become so small that the hard-drive access becomes the dominant factor.

In order to determine the optimal database size for a Fanorona variant, both the solving time and the time required to compute the endgame databases have to be taken into account. Figure 4.9 shows the total time required for creating the endgame databases and for the PN search for the 5×5, 7×5 and 3×9 variant. For the 5×5 variant the optimal database size is 4 pieces. With fewer pieces, the time

Table 4.5: Effectiveness of endgame databases for 5×5 Fanorona.

DB size	0	2	3
DB Time(ms)	-	138	1,228
DB Size	-	600	13,800
Solving Time (ms)	189,919,141	31,610,647	6,176,044
Solving Nodes	101,830,539,576	18,934,832,800	3,546,498,830
DB size	4	5	6
DB Time (ms)	6,547	60,927	326,104
DB Size	171,100	1,593,900	10,980,200
Solving Time (ms)	9,049	2,715	1,033
Solving Nodes	6,323,71	1,509,329	364,278
DB size	7	8	9
DB Time (ms)	2,104,583	8,972,377	41,415,589
DB Size	60,568,200	274,720,050	1,041,917,250
Solving Time (ms)	842	694	577
Solving Nodes	251,236	173,566	119,354

Table 4.6: Effectiveness of endgame databases for 7×5 Fanorona.

DB size	0	2	3
DB Time(ms)	-	141	3,062
DB Size	-	1,190	39,270
Solving Time (ms)	62,371,743	29,159,087	2,300,224
Solving Nodes	39,243,004,473	20,120,149,985	1,685,583,077
DB size	4	5	6
DB Time(ms)	31,350	479,801	3,858,750
DB Size	733,040	9,738,960	100,635,920
Solving Time (ms)	1,108,075	568,325	99,164
Solving Nodes	862,109,429	439,220,855	72,524,679
DB size	7		
DB Time(ms)	77,294,159		
DB Size	847,289,520		
Solving Time (ms)	4,132		
Solving Nodes	1,107,756		

Table 4.7: Effectiveness of endgame databases for 3×9 Fanorona.

DB size	0	2	3	4
DB Time(ms)	-	129	1,836	9,629
DB Size	-	702	17,550	245,700
Solving Time (ms)	39,975	35,379	6,883	12,747
Solving Nodes	33,248,635	28,588,852	5,885,429	10,939,734
DB size	5	6	7	8
DB Time(ms)	104,829	585,177	4,387,274	21,854,833
DB Size	2,421,900	18,352,620	113,891,780	563,899,050
Solving Time (ms)	2,953	1,638	997	2,159
Solving Nodes	2,138,924	1,055,233	233,197	105,343

required for PN search is the critical factor. With more pieces, the time required for computing the endgame databases is the critical factor. The reason for this is that for larger databases, the majority of the positions in the database are not used during the search. For the 7×5 variant, the optimal database size is 5 pieces, and 3 pieces for the 7×5 variant. These optima are at relatively small databases. We also see that the optima occur when the time required for database construction and the time required for solving are of the same order. Taken into account that for 5×9 Fanorona it took 2 to 3 months to construct the databases and slightly more than a week to solve the game, we suspect that the optimal database size for 5×9 Fanorona is 6 or 7 pieces.

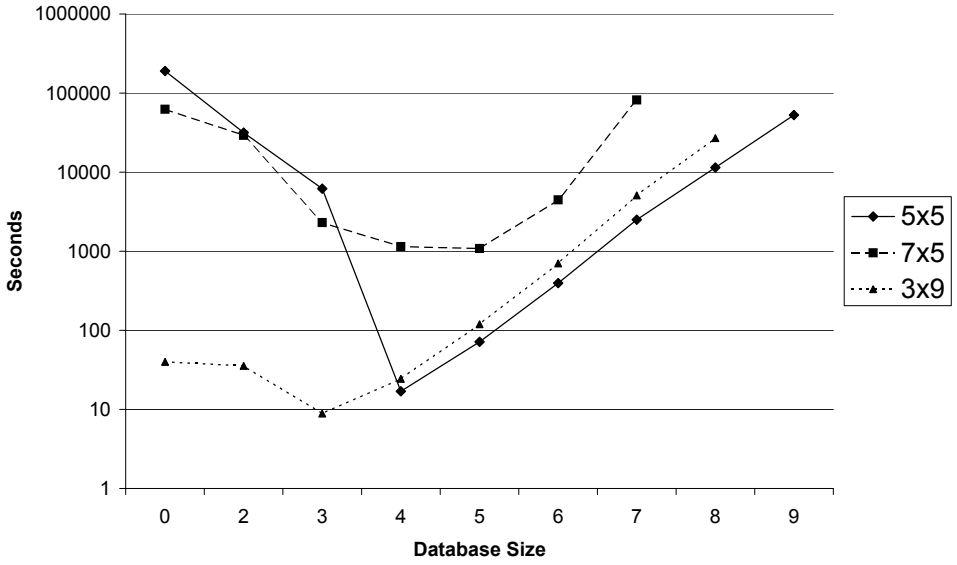


Figure 4.9: Total time required for solving Fanorona variants.

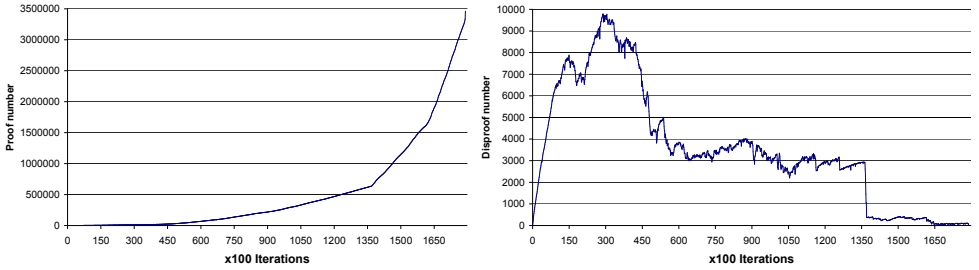
4.6.3 Behavior of the PN Search

The behavior of PN search can be expressed by displaying the development of the proof and disproof number at the root node. During this research, we made an interesting observation. Figure 4.10 and Figure 4.11 show the development of proof and disproof number of the root node during the search. Figure 4.10 depicts a search with the goal that White can win the initial position. Figure 4.11 visualizes a search with the goal that White can at least draw the initial position. We found that the development of the proof and disproof number has a similar pattern for smaller boards and looks like a Chi-Square distribution. The number that goes to zero, independent of its nature (proof or disproof number), always reaches its peak in the beginning and has a long tail at the end. A similar pattern can be found when solving the small boards. For instance, this can be seen when solving the 7×3 board (See Figure 4.12). A reason for this may be that Fanorona is not a *sudden-death* game (i.e., all pieces have to be captured to win the game). When proving the win in the sudden-death game Lines of Action, the development of the proof-number is bell-shaped (Winands, 2008).

4.7 Correctness

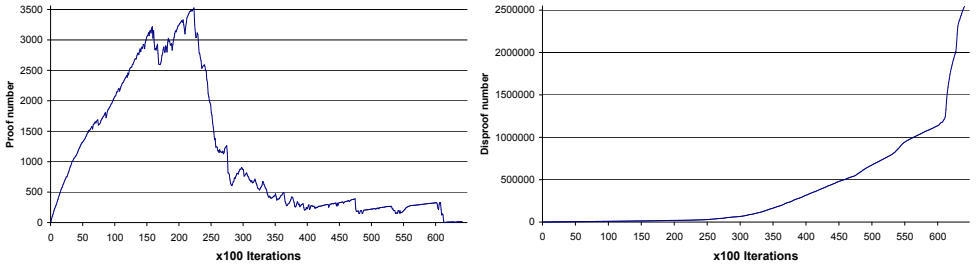
It is important that the obtained results are free of errors. This section gives two arguments why we are convinced that the results presented in this chapter are correct.

First, we verified the code for the retrograde analysis in three ways. (1) We solved



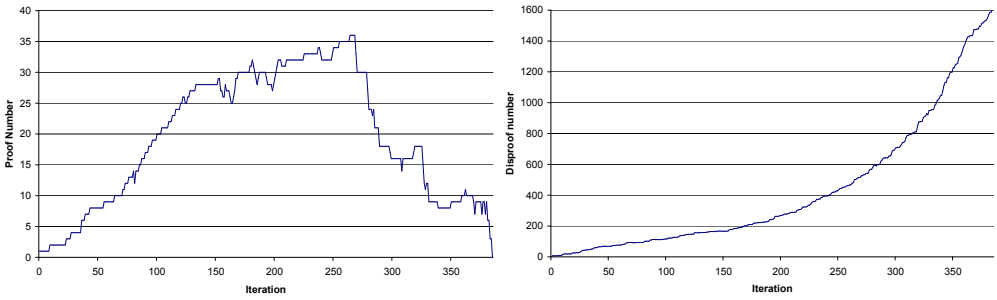
(a) Proof number (b) Disproof number

Figure 4.10: Development of the proof and disproof number when proving that White cannot win the initial 5×9 Fanorona position.



(a) Proof number (b) Disproof number

Figure 4.11: Development of the proof and disproof number when proving that White can at least draw the initial 5×9 Fanorona position.



(a) Proof number (b) Disproof number

Figure 4.12: Development of the proof and disproof number when proving that White can win the initial position on the 7×3 board.

endgame positions with the help of PN search and compared the obtained result with the entry in the database. (2) We did use the database code to construct Lines of Action (LOA) and Surakarta endgame databases, which were verified independently by the programs MIA (Winands, 2004) and SIA (Winands, 2007), respectively. (3) A consistency check was done in order to check for possible bit flips (Schaeffer, 1997). It turns out that no bit flips had occurred.

Second, in order to verify PN search three actions have been taken. (1) The smaller boards were checked manually. This was possible for the 3×3, 5×3, and 3×5 boards. (2) PN search was compared to a standard $\alpha\beta$ search. Non-trivial 5×9 positions were solved by both solvers in order to detect differences. (3) To handle the Graph-History-Interaction (GHI) problem (Palay, 1983; Campbell, 1985), no transposition tables were used during the search. There exist methods that handle the GHI problem with PN search (Breuker *et al.*, 2001a; Kishimoto and Müller, 2004) but these were not implemented.

4.8 Chapter Conclusions and Future Research

Our first and main conclusion of this chapter is that the game of Fanorona (played on the 5×9 board) has been weakly solved and is drawn when both players play optimally. This conclusion was arrived through a well-chosen combination of the proof-number (PN) search variant PN^2 and endgame databases. Combining these two methods is a relatively new approach for solving a game. Simultaneously to our research, checkers was solved with a similar method (Schaeffer *et al.*, 2007; Schaeffer, 2007). Endgame-database statistics show that (1) the player to move has an advantage and (2) that a draw can often be achieved in spite of having fewer pieces than the opponent. Second, we have seen that the optimal endgame database size for the 3×9, 5×5 and 7×5 are 3, 4 and 5 pieces, respectively. The optimal database size is located at the point where the time required for database construction and the time required for solving are of the same order. From this observation, we conclude that the optimum for Fanorona is 6 or 7 pieces. Third, we may conclude that White is able to force a win on board sizes with one side equal to 3. We conjecture that for boards where both sides have at least size 5, White does not have this advantage for the majority of cases (so, we consider 7×5 as an exception because White still wins). The 9×5 board (please note the inversion) of the game has not been fully weakly solved up to now. Preliminary results suggest that this board is much harder to solve than the 5×9 board. We anticipate that the reason for this is the larger distance to the opponent when entering the endgame (see Subsection 4.6.1).

The game-tree and state-space complexity of Fanorona are somewhat higher than those of checkers. Therefore, the following question may arise: why is this game easier to solve than checkers? The answer lies in the properties of the game (e.g., the decision complexity, Allis, Van den Herik, and Herschberg, 1991). In Fanorona, capturing is almost always possible in the opening and middle game; often a large number of pieces is then captured. Thus, the game converges fast to the endgame, where the endgame databases can take over the best-play procedure. The speed of the game convergence is not represented in the game-tree and state-space complexity.

We are confident that if such a measure would be established, it would be higher for Fanorona than for checkers.

In this chapter, Fanorona has been *weakly* solved. We determined a strategy to achieve the game-theoretic value against any opposition starting from the initial position. Solving Fanorona strongly is a challenging subject for future research. At this moment of time (2010) the 9×5 and all larger boards are unsolved. We believe that for solving these board sizes, larger databases are required. It would be interesting to investigate whether the proof and disproof number show a similar pattern as seen in Subsection 4.6.3 when solving other games.

Finally, the time for solving may be reduced significantly by using Evaluation-Function Based Proof-Number Search (EF-PN) (Winands and Schadd, 2011). EF-PN is a general framework for employing a traditional evaluation function in PN search. The search is directed to branches where the evaluation function indicates a promising situation. For Fanorona, the material on the board may be used as an evaluation function.

Chapter 5

Forward Pruning in Chance Nodes

This chapter is an updated and abridged version of the following publication:

1. Schadd, M.P.D., Winands, M.H.M. and Uiterwijk, J.W.H.M. (2009). ChanceProbCut: Forward Pruning in Chance Nodes. *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, P.L. Lanzi, ed., pp. 178−185, IEEE press, Piscataway, NJ, USA.

Human players do not consider the complete game tree to find a good move. Using experience, they are able to prune unpromising variants in advance (forward pruning) (De Groot, 1965). Their game trees are narrow and deep. By contrast, the original minimax algorithm searches the entire game tree up to a fixed depth. Even its efficient variant, the $\alpha\beta$ algorithm (Knuth and Moore, 1975), can only prune safely if a position is proven to be irrelevant to the principal variation (backward pruning). There are several forward-pruning techniques for the $\alpha\beta$ algorithm, such as the null-move heuristic (Beal, 1989; Goetsch and Campell, 1990), (Multi-)ProbCut (Buro, 1995; Buro, 2000), and Multi-Cut (Björnsson and Marsland, 1999; Björnsson and Marsland, 2000; Björnsson and Marsland, 2001), but they cannot guarantee that the best move is not overlooked. These forward-pruning techniques have been applied to deterministic games with perfect information (e.g., Chess, Checkers, and Go), but so far not for non-deterministic or imperfect-information games.

Non-deterministic (stochastic) games introduce an element of uncertainty (Russell and Norvig, 2003), for instance by a roll of dice (e.g., Backgammon and Ludo, Carter, 2007). Besides non-deterministic games with perfect information, there exist deterministic games with imperfect information. Imperfect-information games hide information from the players (e.g., Cluedo (Van Ditmarsch, 2000; Van Ditmarsch, 2001), Scotland Yard (Sevenster, 2006), Stratego (De Boer, 2007) and Kriegspiel (Ciancarini and Favini, 2007)). Both non-deterministic and imperfect-information games are generally more complex than deterministic games of perfect information

(Reif, 1984; Condon, 1992). A game can also be non-deterministic and have imperfect information (Russell and Norvig, 2003) (e.g., games such as Poker (Billings *et al.*, 1998a), Bridge (Smith, Nau, and Throop, 1998) and Risk (Osborne, 2003)). In imperfect-information games, the actual state of the game is not known and a possible state is referred to as a *world*. When performing a tree search in such a game, all possible worlds need to be considered (Frank, 1990; Frank and Basin, 1998). Alternatively, imperfect-information games can be treated as non-deterministic games as if they contain an element of chance (Russell and Norvig, 2003). For example in Stratego, capturing an unknown piece can be treated as a chance event where the chance model is based on the remaining pieces on the board and the knowledge about the opponent. In order to integrate knowledge about the opponent, opponent modeling techniques may be used (Jansen, 1992; Carmel and Markovitch, 1993; Iida *et al.*, 1994; Billings *et al.*, 1998b; Donkers, 2003). These techniques may improve the playing strength (e.g., a win ratio of 55% in Stratego, Stankiewicz and Schadd, 2009).

For non-deterministic games, *expectimax* (Michie, 1966) is the main algorithm of choice. It extends the minimax concept to non-deterministic games, by adding chance nodes to the game tree. Expectimax may be applied to imperfect-information games as well, although alternatives exist (e.g., Partition Search, Ginsberg, 1996). So far, no specific expectimax forward-pruning technique has been designed for these chance nodes.

This chapter answers the fifth research question by proposing *ChanceProbCut*, a forward-pruning technique based on ProbCut (Buro, 1995). ChanceProbCut is able to stop prematurely the search at a chance node in the expectimax framework. This technique estimates values of chance events based on shallow searches. Based on the correlation between evaluations obtained from searches at different depths, ChanceProbCut prunes chance events in advance if the result of the chance node probably falls outside the search window.

The outline of the chapter is as follows. First the expectimax algorithm is explained in Section 5.1. Next, its variants Star1 and Star2 pruning are described in Section 5.2. Thereafter, Section 5.3 discusses forward pruning. We introduce the new forward-pruning technique ChanceProbCut in Section 5.4. Section 5.5 describes the games Stratego, Dice and ChanceBreakthrough, which are used as test domains. The experiments are presented in Section 5.6. Finally, Section 5.7 gives the conclusions and an outlook on future research.

5.1 Expectimax

Expectimax (Michie, 1966) is a brute-force, depth-first game-tree search algorithm that generalizes the minimax concept to non-deterministic games, by adding *chance nodes* to the game tree (in addition to MIN and MAX nodes). A chance node is added when a die roll occurs (non-deterministic) or information has to be revealed (imperfect information). At chance nodes, the heuristic value of the node (or expectimax value) is equal to the weighted sum of the heuristic values of its successors. For a state s, its expectimax value is calculated by the following formula:

$$expectimax(s) = \sum_i P(c_i) \times V(c_i) \tag{5.1}$$

where c_i represents the ith child of s, $P(c)$ is the probability that child c will be reached, and $V(c)$ is the value of child c.

We explain expectimax in the following example. Figure 5.1 depicts an expectimax tree. In the figure squares represent chance nodes, regular triangles MAX nodes and inversed triangles MIN nodes. In Figure 5.1, node A corresponds to a chance node with two possible events, after which it is the MIN player's turn. The value of node A is calculated by weighting the outcomes of both chance events. In this example, $expectimax(A) = 0.9 \times -250 + 0.1 \times 250 = -200$.

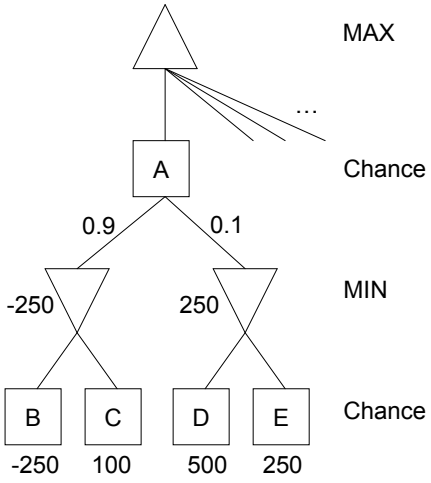


Figure 5.1: An example expectimax tree.

When searching the children of a chance node, the values of the children might be below or above the search window. However, it is possible that the combination of these values falls inside the search window. Therefore, the search window needs to be reset at every child of a chance node. Figure 5.2 shows an example. The value of chance node A is $0.6 \times 2 + 0.4 \times 10 = 5.2$, which is inside the window. If the $\alpha\beta$ search window (4, 6) is passed down to the first MIN node, the value of the first MIN node will be 4, instead of 2. Passing down the search window to the second MIN node has no effect in this case. However, the value of the chance node is incorrectly calculated as $0.6 \times 4 + 0.4 \times 10 = 6.4$ (correct is 5.2). This could even cause an incorrect pruning at the MAX node above A.

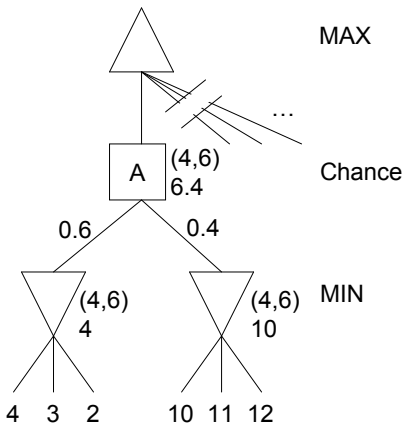


Figure 5.2: Search windows at chance nodes cause incorrect pruning.

5.2 Pruning in Chance Nodes

As stated by Hauk, Buro, and Schaeffer (2006a), the basic idea of expectimax (Ballard, 1983) is sound but slow. Star1 and Star2 exploit a bounded heuristic evaluation function to generalize the $\alpha\beta$ pruning technique to chance nodes (Ballard, 1983; Hauk *et al.*, 2006a). $\alpha\beta$ pruning imposes a search window (α, β) at each MIN or MAX node in the game tree. Remaining successor nodes can be pruned as soon as the current node's value is proven to fall outside the search window. Star1 and Star2 apply this idea to chance nodes. The difference with $\alpha\beta$ is that the search cannot stop at a chance node as soon as one successor falls outside the search window. To end the search, the weighted sum of all successors has to fall outside the search window. These techniques can be applied to non-deterministic games (Ballard, 1983; Hauk, 2004) and imperfect-information games (Stengård, 2006).

5.2.1 Star1

Star1 is able to create cutoffs if the lower and upper bound of the evaluation function are known (called L and U) (Ballard, 1983; Hauk, 2004). These bounds are the game-theoretic values of terminal positions ($Loss$ and Win, respectively). If we have reached the ith successor of a chance node, after having searched the first $i-1$ successors and obtained their values, then we can obtain bounds for the value of the chance node. A pruning takes place if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + L \times (N - i)}{N} \geq \beta \tag{5.2}$$

or

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + U \times (N - i)}{N} \leq \alpha \tag{5.3}$$

where V_i is the value of node i and N the number of children, assuming that each child occurs with a probability $\frac{1}{N}$.

A lower bound is obtained by assuming that all remaining successors return L, an upper bound is obtained by assuming that all remaining successors return U. Safe pruning can be performed if the range defined by these bounds falls outside the search window. Instead of computing these bounds from scratch, it is possible to use an update rule (Ballard, 1983; Hauk, 2004).

Figure 5.3 demonstrates this pruning technique. The evaluation function is bound to the interval [-10, 10] in this example. Assume that node A is entered with an $\alpha\beta$ window of (5, 10). When the third child is searched, the theoretic upper bound for the chance node is $0.25 \times (2 + 4 - 8) + 0.25 \times 10 = 2$. Thus, even if the last child would return the maximum value, the value of node A will always be lower than α (i.e., 5). Thus, there is no need to search the last child.

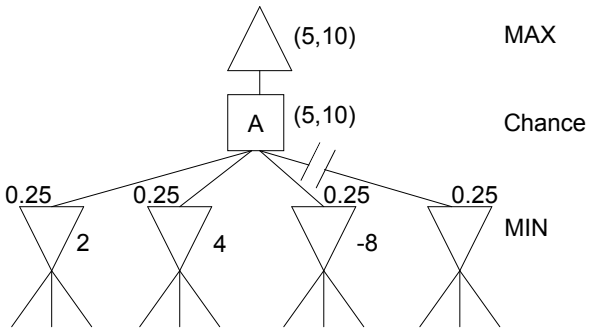


Figure 5.3: Successful Star1 pruning.

Star1 also enables the possibility to compute a window for the children of a chance node, which was previously not possible. The window is defined by the value which the next child should have, so that Formulas 5.2 and 5.3 fall outside the search window. Usually, only the last children to be visited benefit from this enhancement.

5.2.2 Star2

While Star1 returns the same value as expectimax, and uses fewer node expansions to obtain the same value, the amount of search reduction is generally not impressive (Hauk, Buro, and Schaeffer, 2006b). This is due to Star1's pessimistic nature. To obtain more accurate bounds for a node, Star2 probes each child (Ballard, 1983; Hauk, 2004) during a probing phase, which precedes the regular search phase. By only searching one of the available opponent moves, a bound for this node is obtained. This is a lower bound for a MAX node, and an upper bound for a MIN node. This bound is then backpropagated for calculating a more accurate bound for the chance node. A safe pruning is performed at a chance node followed by MAX nodes if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + (W_{i+1} + \ldots + W_N)}{N} \geq \beta \tag{5.4}$$

where W_i is the probed value for child i. At a chance node followed by MIN nodes, a pruning takes place if

$$\frac{(V_1 + V_2 + \ldots + V_{i-1}) + V_i + (W_{i+1} + \ldots + W_N)}{N} \leq \alpha \tag{5.5}$$

Formulas 5.4 and 5.5 assume that each child occurs with probability $\frac{1}{N}$, where N is the number of children.

We explain Star2 in the following example. Figure 5.4 depicts an expectimax tree with Star2 pruning. Node A is reached with an $\alpha\beta$ window of $(-150, 150)$. At this point, the theoretic lower and upper bounds of node A are $[-1000, 1000]$, which correspond to losing and winning the game. Star2 continues with probing the first possible chance event (i.e., investigating node B only). The result of this probe produces an upper bound for this chance event (≤ -250). The theoretic upper bound for A is now updated according to the expectimax procedure: $-250 \times 0.9 + 1000 \times 0.1 = -125$. A cut is not possible yet and the same procedure is applied to the second chance event. After the second probe, the theoretic range of A is $[-1000, -175]$ which is outside the $\alpha\beta$ window. Now nodes C and E can be pruned. Additional search effort would be caused if no pruning would occur. This effect can be nullified by using a transposition table. In case probing does not generate a cutoff, the probed values can be used for pruning or tightening the window during the regular search.

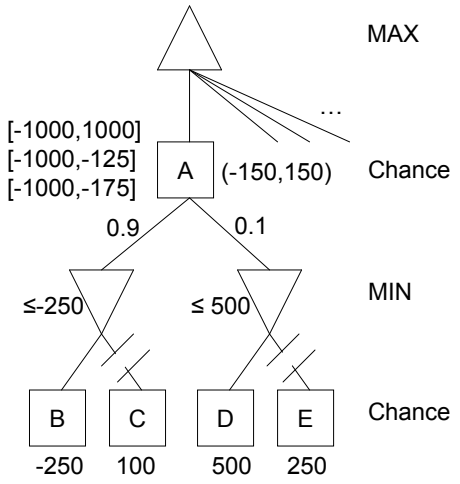


Figure 5.4: Successful Star2 pruning in the probing phase.

5.3 Forward Pruning

Human players are able to find good moves without searching the complete game tree. Using experience they are able to prune unpromising variations in advance (De Groot, 1965). Human players also select promising variants and search them deeper. In $\alpha\beta$ search this concept is known as *variable-depth search* (Marsland and Björnsson, 2001). The technique to abandon some branches prematurely is called *forward pruning*. The technique to search certain branches beyond the nominal depth is called *search extensions*. As such, the search can return a different value than a fixed-depth search.

In the case of forward pruning, the complete minimal game tree may not expanded (Björnsson, 2002), and good moves may be overlooked. However, the rationality is that although the search occasionally goes wrong, the time saved by pruning non-promising playing lines is generally better used to search other lines deeper, i.e., the search effort is concentrated where it is more likely to benefit the quality of the search result.

The real task when doing forward pruning is to identify move sequences that are worth considering more closely, and others that can be pruned with minimal risk of overlooking a good continuation. Ideally, forward pruning should have low risk, limited overhead, be applicable often, and be domain independent. Usually, improving one of these factors will worsen the others (Björnsson, 2002).

The null-move heuristic (Beal, 1989; Goetsch and Campell, 1990) is a well-known approach of forward pruning. Forfeiting the right to move is called a *null move*. This might be a legal move to play (e.g., Go), but in many games it is an illegal move (e.g., Chess). Instead of searching a position to depth d, the null-move is searched to depth $d-R$ (typically $R \in \{2,3\}$). If the result of this search is greater than β, a cutoff is made. The intuition is that generally making a move is better than passing (except in *Zugzwang* positions). Thus, if passing creates a cutoff, it is likely that a regular move creates a cutoff as well. The idea of using null moves in the search tree has been known for a long time (Adelson-Velskiy, Arlazarov, and Donskoy, 1975), and is nowadays used by most chess programs.

Multi-Cut pruning is another forward-pruning technique (Björnsson and Marsland, 1999; Björnsson and Marsland, 2001). Before examining an expected CUT node (cf. Section 2.3) to full depth, the first M child nodes are searched at a reduced depth $d-R$. If at least C child nodes fall outside the search window, a cutoff is produced. In general the behavior of Multi-Cut is as follows. The higher M and R are and the lower C is, the higher the number of prunings is. This technique was generalized later to ALL nodes (Winands *et al.*, 2005).

The *ProbCut* heuristic (Buro, 1995) uses shallow searches to predict the result of deep searches. A branch is pruned if the shallow search produces a cutoff, with a certain confidence bound. This heuristic works well for techniques where the score of a position does not significantly change when searched deeper, as in Othello. The technique was further enhanced as the *Multi-ProbCut* heuristic (Buro, 2000). Multi-ProbCut performs multiple shallow searches of different depths to decide whether a subtree is pruned.

All these heuristics are applicable at MIN and MAX nodes. However, not much

work has been done on forward pruning in trees with chance nodes. Smith and Nau (1993) set up a theoretic model of forward pruning in binary trees with chance nodes. No forward-pruning techniques for chance nodes in non-binary trees have been proposed so far.

5.4 ChanceProbCut

So far, it has not been investigated whether forward pruning can be beneficial at chance nodes. The null-move and Multi-Cut heuristic cannot be adapted to chance nodes because these techniques are based on applying moves. We introduce ChanceProbCut to forward prune unpromising chance nodes. This technique is inspired by ProbCut (Buro, 1995), and uses this idea to generate cutoffs at chance nodes. A shallow search of depth $d - R$ is an indicator of the true expectimax value v_d for depth d. Now, it is possible to determine whether v_d would produce a cutoff with a prescribed likelihood. If so, the search is terminated and the appropriate window bound is returned. If not, a regular search is performed.

ChanceProbCut adapts the ProbCut idea to alter the lower and upper bounds for each possible event at a chance node. A search with reduced depth is performed for each chance event. The result of this search, v_{d-R}, is used for predicting the value of the chance event for depth d (v_d). The prediction of v_d is calculated by confidence bounds in a linear regression model (Buro, 1995). The bounds can be estimated by:

$$ELowerBound(v_d) = v_{d-R} \times a + b - t \times \sigma \tag{5.6}$$

and

$$EUpperBound(v_d) = v_{d-R} \times a + b + t \times \sigma \tag{5.7}$$

where a, b and σ are computed by linear regression, and t determines the size of the bounds. Using these bounds, it is possible to create a cutoff if the range of the chance node, computed by multiplying the bounds of each child with the corresponding probabilities, falls outside the search window.

At a chance node, variables containing the lower and upper bounds are updated during each step. A cutoff may be obtained with values computed from different techniques. It is possible that during the regular search a cut is produced using a combination of transposition table probing (Veness and Blair, 2007), ChanceProbCut, Star2 and exact values, because the bounds are updated when new information becomes available.

When the regular search with depth d is started, the bounds obtained by ChanceProbCut can be used as a search window. It is unlikely that v_d falls outside this interval. We note that it is possible to perform a re-search after the regular search returns with a value outside the search window. However, we did not implement this because an error only partially contributes to the value of the chance node.

An example of how ChanceProbCut prunes is given in Figure 5.5. Here, the regression model $v_d = v_{d-R}$ is assumed, with confidence interval 50. The first ChanceProbCut search returns the bounds [300, 400] for the first child, changing

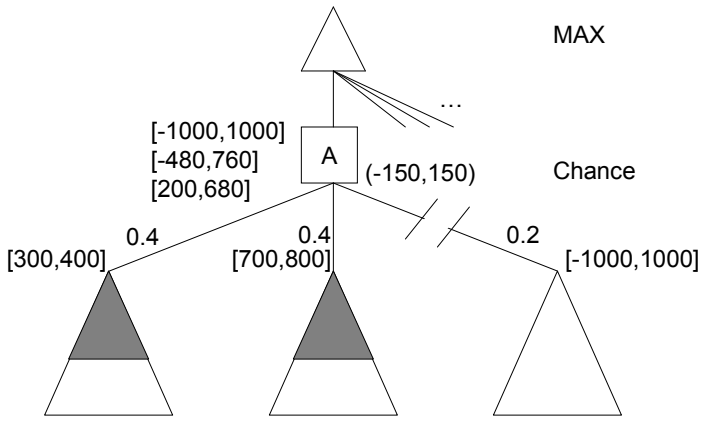


Figure 5.5: ChanceProbCut pruning.

the lower bound of the chance node to $300 \times 0.4 - 1{,}000 \times 0.6 = -480$ and the upper bound to $400 \times 0.4 + 1{,}000 \times 0.6 = 760$. The second ChanceProbCut search returns the window [700, 800]. Now, the lower bound of the chance node is computed as $300 \times 0.4 + 700 \times 0.4 - 1{,}000 \times 0.2 = 200$ and the upper bound is computed as $400 \times 0.4 + 800 \times 0.4 + 1{,}000 \times 0.2 = 680$. The range of the chance node, [200, 680], falls outside the search window and the next node is pruned.

Figure 5.6 depicts a second example, in which ChanceProbCut fails to prune any nodes. The search finds the same values for the first child as in the previous example. The next search reveals [0, 100] as bounds for the second child. This time, it is not possible to prune (with bounds [-80, 400] for the chance node). Even after the next search produces the bounds [50, 150] for the third child, the range of the chance node, [130, 230], does not fall outside the search window. However, after the regular search returns 400 for the first child, the search is terminated based on previously estimated ChanceProbCut values.

Finally, the pseudo code is shown in the next two algorithms. They are described in the negamax framework. Algorithm 5.1 describes the $\alpha\beta$ search part. Here it is assumed that after each move follows a chance node (Line 9). Algorithm 5.2 shows the pseudo code for a chance node. After initialization of the arrays for the bounds (Line 3−6), ChanceProbCut is used (Line 8−20). The confidence interval around the obtained value is computed in Line 13−14, based on linear regression (Buro, 1995). In the case that ChanceProbCut does not produce a pruning, the Star2 search is started (Line 22−32), which might improve the estimates obtained by ChanceProbCut. The *Probe()* procedure (Line 24) is used for obtaining an upper bound of the chance node (Hauk *et al.*, 2006a; Ballard, 1983). If Star2 also fails to obtain a pruning, the algorithm continues with the regular Star1 search (Line 34−44), which benefits from the bounds calculated earlier.

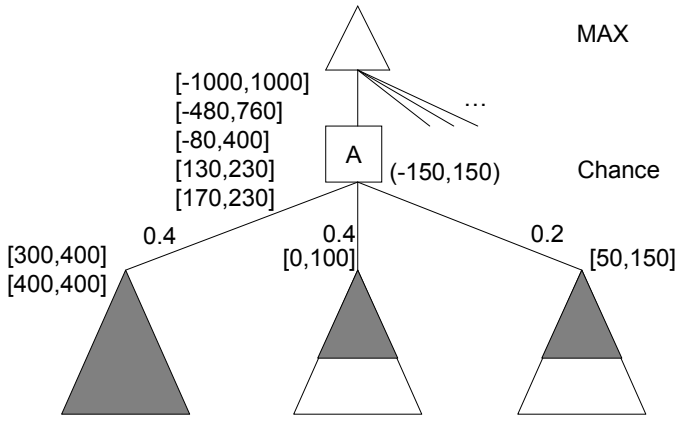


Figure 5.6: The regular search prunes with help of ChanceProbCut.

5.5 Test Domain

To test whether ChanceProbCut performs well, we use three games, Stratego, Dice, and ChanceBreakthrough.

5.5.1 Stratego

Stratego is a deterministic imperfect-information game. It was invented at least as early as 1942 by Mogendorff. The game was sold by the Dutch publisher *Smeets*

```
1: Search(alpha, beta, depth)
2:
3: if depth==0 then
4:   return eval()
5: end if
6:
7: for all Moves i do
8:   doMove(i);
9:   value = −ChanceNode(−beta, −alpha, depth−1);
10:  undoMove(i);
11:  if value ≥ beta then
12:    return beta;
13:  end if
14:  alpha = max(alpha, value);
15: end for
16:
17: return alpha;
```

Algorithm 5.1: $\alpha\beta$ search part of expectimax.

```
1: ChanceNode(alpha, beta, depth)
2:
3: for all ChanceEvents i do
4:    lowerBound[i] = ELowerBound[i] = L;
5:    upperBound[i] = EUpperBound[i] = U;
6: end for
7:
8: if depth > R then //ChanceProbCut
9: for all ChanceEvents i do
10:   doMove(i);
11:   v = Search(L, U, depth−1−R)
12:   undoMove(i);
13:   ELowerBound[i] = max(L, A × v + B − STDEV × t);
14:   EUpperBound[i] = min(U, A × v + B + STDEV × t);
15:   if ∑_j P_j× ELowerbound[j] ≥ beta then return beta;
16:   end if
17:   if ∑_j P_j× EUpperbound[j] ≤ alpha then return alpha;
18:   end if
19: end for
20: end if
21:
22: for all ChanceEvents i do //Star2
23:    doMove(i);
24:    v = Probe(lowerBound[i], upperBound[i], depth−1);
25:    undoMove(i);
26:    upperBound[i] = max(upperBound[i], v);
27:    EUpperBound[i] = min(upperBound[i], EUpperBound[i]);
28:    if upperBound[i] < ELowerBound[i] then ELowerBound[i] = L;
29:    end if
30:    if ∑_j P_j× upperbound[j] ≤ alpha then return alpha;
31:    end if
32: end for
33:
34: for all ChanceEvents i do //Regular Search with Star1
35:    doMove(i);
36:    v = Search(min(Star1LowerBound(), ELowerBound[i]),
37:         max(Star1UpperBound(), EUpperBound[i]), depth−1);
38:    undoMove(i);
39:    lowerBound[i] = upperBound[i] = v;
40:    if ∑_j P_j× lowerbound[j] ≥ beta then return beta;
41:    end if
42:    if ∑_j P_j× upperbound[j] ≤ alpha then return alpha;
43:    end if
44: end for
45:
46: return ∑_i P_i× lowerbound[i]
```

Algorithm 5.2: ChanceProbCut for chance nodes.

and Schippers between 1946 and 1951 (District Court of Oregon, 2005). In this subsection, we first describe briefly the rules of the game, then give the game-tree and state-space complexity and thereafter present related work.

Rules

The following rules are an edited version of the Stratego rules published by the Milton Bradley Company in 1986 (Milton Bradley Co., 1986). Stratego is played on a 10×10 board. The players, White and Black, place each of their 40 pieces in such a way that the back of each piece faces the opponent in a 4×10 area. The movable pieces are divided in ranks (from the lowest to the highest): Spy, Scout, Miner, Sergeant, Lieutenant, Captain, Colonel, Major, General, and Marshal. Each player has two types of unmovable pieces, the Flag and the Bomb. An example initial position is depicted in Figure 5.7. The indices represent the ranks, where the highest rank has index 1 (the Marshal), and all decreasing ranks have increasing indices (exceptions are S=Spy, B=Bomb and F=Flag).

4	8	B	F	B	8	6	9	8	3
6	7	2	B	7	9	S	1	B	7
3	B	5	4	6	8	B	8	9	9
5	9	6	9	9	5	9	4	7	5
9	6	7	3	9	6	4	5	6	9
5	9	8	9	6	9	5	2	B	9
B	7	S	1	8	B	3	7	4	5
F	B	8	4	B	7	B	8	9	8

Figure 5.7: A possible initial position in Stratego.

Players move alternately, starting with White. Passing is not allowed. Pieces are moved to orthogonally-adjacent vacant squares. The Scout is an exception to this rule, and must be moved like a rook in chess. The *Two-Squares Rule* and the *More-Squares Rule* prohibit moves which result in repetition.[1] The lakes in the center of the board contain no squares; therefore a piece can neither move into nor cross the lakes. Only one piece may occupy a square.

[1]For details of these rules we refer to the International Stratego Federation (ISF), www.isfstratego.com.

A piece, other than a Bomb or a Flag, may attempt to capture an orthogonally adjacent opponent's piece; a Scout may attempt to capture from any distance. When attempting a capture, the ranks are revealed and the weaker piece is removed from the board. The stronger piece is positioned on the square of the defending piece. If both pieces are of equal rank, both are removed. The Flag is the weakest piece, and can be captured by any moving piece. The following special rules apply to capturing. The Spy defeats the Marshal if it attacks the Marshal. Each piece, except the Miner, is removed from the board when attempting to capture a Bomb.

The game ends when the Flag of one of the players is captured. The player whose Flag is captured loses the game. A player also loses the game if there are no legal moves. The game is drawn if both players cannot move.

Game-Tree and State-Space Complexity

Based on a database of 4,500 human games, the average game length is estimated to be 318 plies with an average branching factor of 22. During a game, there are on average 30 captures of unknown pieces with on average 7 possible chance events (Arts, 2010). Thus, the game-tree complexity is $22^{318} \times 7^{30} \approx 10^{452}$. The state-space complexity of Stratego is computed by the following formula.

$$
\begin{aligned}
40 \times 40 \times \left(\sum_{\substack{red\\bombs}}^{y} \sum_{\substack{blue\\bombs}}^{z} \frac{39!}{(39-y)! \times y!} \times \frac{39!}{(39-z)! \times z!} \right) \times \\
\sum_{\substack{red\\spies}}^{a} \sum_{\substack{red\\scouts}}^{b} \cdots \sum_{\substack{blue\\generals}}^{s} \sum_{\substack{blue\\marshals}}^{t} \left(\frac{(90-y-z)!}{(90-y-z-a)! \times a!} \times \right. \\
\left. \frac{(90-y-z-a)!}{(90-y-z-a-b)! \times b!} \times \cdots \times \frac{(90-y-z-a-b-\cdots-s)!}{(90-y-z-a-b-\cdots-s-t)! \times t!} \right)
\end{aligned}
\tag{5.8}
$$

The first line takes care of the locations of the Flag and the Bombs, while the other two lines compute all possible positions of all other pieces. The number of free squares depends on the total number of pieces on the board. There are 92 (where 2 of them are always occupied by both the Flags) available squares minus the number of pieces on board. The upper bound of the state-space of Stratego is 10^{115}.

Previous Work

Stratego has received some scientific attention in the past. Treijtel and Rothkrantz (2001) created a player based on multi-agent negotiations. Stengård (2006) investigated different search techniques for this game. De Boer, Rothkrantz, and Wiggers (2008) described the development of an evaluation function using an extensive amount of domain knowledge in a 1-ply search. Schadd and Winands (2009) tested Evaluation-Based Quiescence Search in Stratego. At this moment, computers play Stratego at an amateur level (Satz, 2008). An annual Stratego computer tourna-

ment[2] is held on Metaforge with an average of six entrants (Satz, 2008; Schadd and Satz, 2008; Jug and Schadd, 2009).[3] Finally, we remark that some research has been done in Siguo, a four-player variant of Stratego (Xia *et al.*, 2005; Xia *et al.*, 2007; Lu and Xia, 2008).

5.5.2 Dice

The game of Dice is a two-player non-deterministic game with perfect information, recently invented by Hauk (2004), in which players take turns placing checkers on an $m \times m$ grid. One player plays columns, the other plays rows. Before each move, an m-sided die is rolled to determine the row or column into which the checker must be placed. The winner is the first player to achieve a line of m checkers (orthogonally or diagonally).

Based on 1,000 computer-played games for the 11×11 variant, the average game length is 78 plies with an average branching factor of 7. Each move is preceded by a roll of the dice, which can have 11 chance events. Thus, the game-tree complexity is $7^{78} \times 11^{78} \approx 10^{147}$. The state-space complexity is $3^{121} \approx 10^{58}$, because there a point in 11×11 grid can be empty, or be occupied by a white or black checker.

The advantages of this game are that (1) it is straightforward to implement and (2) that many chance nodes exist. A disadvantage is that the result of a Dice game is partially dependent on luck. A deep search is still beneficial. Hauk (2004) showed that a 9-ply player wins 65% against a 1-ply player.

Hauk (2004) used Dice to demonstrate the pruning effectiveness of Star-minimax algorithms in a non-deterministic game. Moreover, Veness and Blair (2007) used it to test StarETC, a variant of Enhanced Transposition Cutoffs (Schaeffer and Plaat, 1996).

5.5.3 ChanceBreakthrough

The game of ChanceBreakthrough is the non-deterministic variant of the deterministic Breakthrough game. The standard Breakthrough game was invented by Dan Troyka in 2001 and is played on an 8×8 checkers board (Handscomb, 2001). The pieces are set up in the two back ranks, Black at top and White at the bottom, as can be seen in Figure 5.8. White goes first and then players alternately move. A piece may move one space straight or diagonally forward if the target square is empty. However, it can only capture diagonally like a pawn in chess. The goal of the game is to "breakthrough" and reach the other side of the board before the opponent can achieve it. The game has been used to test Gibbs Sampling (Björnsson and Finnsson, 2009) and to learn search extensions (Skowronski, Björnsson, and Winands, 2010).

In ChanceBreakthrough each player has an extra type of move available at every turn: rolling the dice. Two eight-sided dice are rolled, one representing the rows, the other representing the columns. The resulting square on the board will be emptied. This move is preferable in deadlock situations or when the opponent player has

[2] http://www.strategousa.org/wiki/index.php/Main_Page
[3] http://www.metaforge.net/

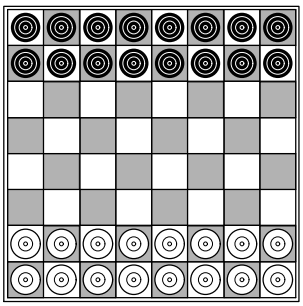


Figure 5.8: Initial position of ChanceBreakthrough.

more pieces than one self (i.e., the probability is higher to remove an opponent piece). Based on 1,000 selfplay games with 5 seconds per move, the game length is on average 53 plies and the average branching factor is 24. There are on average 14 possible chance events. The game-tree complexity is $(24 + 14)^{53} \approx 10^{84}$. The last row on each player's side is "private", because the game ends if an opponent would occupy a square on it. Taking this into account, we computed the state-space complexity of ChanceBreakthrough to be approximately 10^{25}, in a similar manner as Formula 5.8.

5.5.4 Game Engines

We implemented an expectimax engine for Stratego, Dice, and ChanceBreakthrough, enhanced with the Star1 and Star2 pruning algorithms (Ballard, 1983; Hauk, 2004). Furthermore, the history heuristic (Schaeffer, 1983), the killer heuristic (Akl and Newborn, 1977), transposition tables (Greenblatt *et al.*, 1967; Slate and Atkin, 1977) and StarETC (Veness and Blair, 2007) are used.

Stratego Engine

A well-performing evaluation function is an important factor for the playing strength of a Stratego program. For our program, we have chosen a material-based approach, partially based on research by De Boer (2007). The evaluation function is bound to [-1,000, 1,000] which corresponds to losing or winning the game. The evaluation function of a node is subtracted by the evaluated value of the root node. Because the evaluation function is bounded to [-1,000, 1,000] and the sum of the values of the pieces can exceed this interval, a bias is included to find a good move in positions where the evaluations of all leaf nodes would fall outside [-1,000, 1,000]. The piece values are shown in Table 5.1.

The Spy has a value of 100, but when the opponent's Marshal is captured, the value of the Spy is reduced to 10. Furthermore, the value of a Miner is dependent on the knowledge regarding the bombs, based on De Boer (2007). Depending on the game situation regarding the Flag of the opponent and the number of Miners left, a Miner may be valued as high as 200 points or as low as 10 points. Also, each square on the board is assigned a value of importance (i.e., increase the distance of

Table 5.1: Stratego piece values.

Spy	Scout	Miner	Sergeant	Lieutenant	Captain
100(10)	10	100*	20	50	100
Major	Colonel	General	Marshal	Bomb	Flag
140	175	300	400	75	1,000

the opponent to the own Flag, while decreasing the own distance to the opponent's Flag). The importance value of a square can be up to 50 points. A bonus is given if information of a piece is hidden. This bonus is set to 30% of the value of the piece. Furthermore, a variable counts the number of subsequent moves without capturing. For each two non-capture moves, the evaluation score is reduced by 1 point. Using this factor, more risk is taken when nothing happens in the game. The evaluation function is equipped with Evaluation Based Quiescence Search (EBQS) (Schadd and Winands, 2009). EBQS is able to estimate the quiescent value of a position without performing an actual quiescence search, which can result in a search explosion for Stratego. Removing large fluctuations in the evaluation function leads to a better prediction of deeper searches, and thus to better forward pruning. A small random factor (half a Scout) is included to prevent that games are (partially) repeated in the selfplay experiments.

When capturing occurs, a chance node is added to the tree. The probabilities of each event are based on the events in the past (i.e., piece movement, captured pieces). For instance, the opponent has eight unknown pieces left: 1 Flag, 1 Bomb, 1 General, 2 Colonels and 3 Captains. When attempting to capture a piece, the probabilities of encountering a General, Colonel and Captain are $\frac{1}{8}$, $\frac{2}{8}$, and $\frac{3}{8}$, respectively. If the piece has moved in the past, it cannot be a Flag or Bomb. In this case, the probabilities of encountering a General, Colonel and Captain are $\frac{1}{6}$, $\frac{2}{6}$, and $\frac{3}{6}$, respectively. The current approach does not lead to mixed strategies which could be necessary to conceal information. However, a possibility is to alternate the assignment of probabilities for the chance events to become less predictable.

Dice Engine

The evaluation function counts the number of checkers which can be used for forming lines of size m. Checkers, which are fully blocked by the opponent, are not counted. Partially blocked checkers get a lower value. The evaluation function is bound to the interval $[-10, 10]$. A small random factor ($\frac{1}{10}$ of a checker) is added, as well.

ChanceBreakthrough Engine

In Breakthrough, the most important factor is the number of pieces on the board. With more pieces it is easier to break through the lines of the opponent. In ChanceBreakthrough this is the most important factor as well, valued 200 points per piece. Furthermore, a progression factor was added to the evaluation function, rewarding 50 points for each row of the most advanced piece. The evaluation function is bound to the interval $[-1500, 1500]$. A random factor of 5 points was included.

5.6 Experiments and Results

In this section, we first discuss the results of ChanceProbCut in the game of Stratego. Second, we test ChanceProbCut in the game of Dice. Third, ChanceProbCut is evaluated in the game of ChanceBreakthrough. All experiments were performed on an AMD64 2.4 GHz computer.

5.6.1 Stratego

This subsection presents all the results obtained in the domain of Stratego.

Determining Parameters

The first parameters to choose are the depth reduction R and the depths d at which ChanceProbCut is applied. The game tree in Stratego is not regular, meaning that not always a chance node follows a MIN/MAX node. Due to this we do not count chance nodes as a ply for Stratego. While in theory this technique can be applied at each search depth, we limit the applicability to $d \in \{4, 5\}$. R is set to 2, because otherwise an odd-even effect might occur. To find the parameters σ, a, and b for the linear regression model, 500 value pairs (v_{d-R}, v_d) have been determined. These value pairs are obtained from 200 begin, 200 middle, and 100 endgame positions, created using selfplay.[4] In Figure 5.9 the model is shown for depths 2 and 4. Figure 5.10 shows the linear regression model for depths 3 and 5. v_{d-R} is denoted on the x-axis; v_d is denoted on the y-axis. Both figures show that the linear regression model is able to estimate the value of v_d with a reasonable standard deviation.

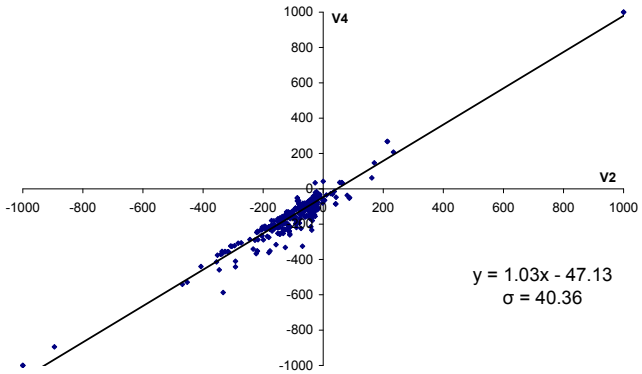


Figure 5.9: Evaluation pairs at depths 2 and 4 in Stratego.

[4] A position was stored after 50, 150 and 300 moves for begin, middle and endgame, respectively. All test positions for Stratego, Dice and ChanceBreakthrough can be downloaded from http://www.personeel.unimaas.nl/Maarten-Schadd/TestSets.html.

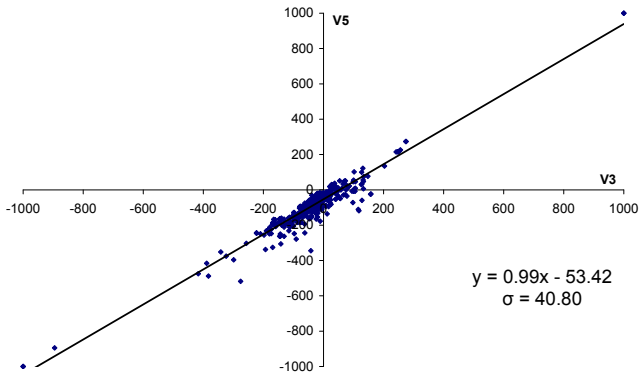


Figure 5.10: Evaluation pairs at depths 3 and 5 in Stratego.

Table 5.2: Performance of ChanceProbCut at depth 7 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	55,207,171	-	-	-
12.8	45,293,207	17.6%	0.14	1.68
6.4	38,748,056	29.8%	0.26	2.88
3.2	34,483,433	37.5%	0.43	4.94
1.6	32,907,389	40.4%	0.43	4.94
0.8	30,661,362	44.5%	0.44	4.94
0.4	30,220,785	45.3%	0.67	7.32
0.2	30,149,981	45.4%	0.73	7.45
0.1	29,950,276	45.7%	0.73	7.45
0.05	29,922,575	45.8%	0.73	7.45

Tuning Selectiveness

Next, we have to find the optimal value for t. If a too large value for t is chosen, the regular search will always be performed. The reduced-depth searches will just cause an overhead. If a too small value for t is chosen, the search might return incorrect values. For tuning this parameter, we look at the reduction of the game tree and the quality of the returned move. For this experiment, the regression models from Figure 5.9 and 5.10 are used at depths 4 and 5, respectively. 500 positions consisting of 200 begin, 200 middle, and 100 endgame situations were tested. Tables 5.2, 5.3 and 5.4 give the results of tuning the t parameter for depths 7, 9 and 11, respectively. For these experiments, the random factor was turned off.

In these tables, *ACPCME* is the average ChanceProbCut move error. This measure compares the value of the best move found by ChanceProbCut with the actual value of this move in a full search. *DCPCME* is the standard deviation of the ChanceProbCut move error.

In Tables 5.2, 5.3, and 5.4 we observe that it is possible to reduce the size of the

Table 5.3: Performance of ChanceProbCut at depth 9 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	1,458,546,577	-	-	-
12.8	586,014,017	59.8%	0.43	3.84
6.4	357,070,424	75.5%	0.24	2.23
3.2	241,811,100	83.4%	0.56	2.83
1.6	215,023,725	85.3%	0.82	3.36
0.8	193,707,382	83.7%	1.16	4.27
0.4	189,763,738	87.0%	1.39	4.87
0.2	188,827,461	87.1%	1.49	5.08
0.1	187,542,414	87.1%	1.51	5.09
0.05	182,938,163	87.5%	1.51	5.11

Table 5.4: Performance of ChanceProbCut at depth 11 for 500 Stratego positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	33,251,941,846	-	-	-
12.8	12,140,806,418	63.5%	5.60	32.11
6.4	8,221,121,960	75.3%	12.56	56.46
3.2	6,305,673,987	81.0%	14.76	57.70
1.6	5,151,596,173	84.5%	16.26	58.21
0.8	4,796,794,009	85.6%	17.44	58.53
0.4	4,546,881,315	86.3%	17.71	58.55
0.2	4,477,761,845	86.5%	17.98	58.58
0.1	4,789,667,704	85.6%	18.04	58.60
0.05	4,763,428,798	85.7%	18.11	58.60

tree significantly without a loss of quality. In Table 5.2 we see that the tree can be reduced by 44.5% before the ChanceProbCut move is wrong by more than 0.5 point on average (i.e., $\frac{1}{20}$ of a Scout). At depth 9, the tree can be reduced with 75.5% of its size without almost no error at all. At depth 11, we observe that a more carful setting is needed. With t set to 12.8, the tree can be reduced by 63.5% with an average evaluation error of 5 points (half a Scout).

When performing the experiments, we noticed that when the t parameter is decreased, the error grows, resulting in a larger error of the best move, and a larger deviation. We also observed that at a larger depth, a more careful t is required. At larger depths ChanceProbCut is applied more often and the error rate increases.

Selfplay

For forward-pruning techniques, a reduction of nodes searched cannot be seen as an indicator of improvement. Selfplay experiments have to be played in order to examine whether ChanceProbCut improves the playing strength.

We decided to test ChanceProbCut with one second per move, typically reaching a search depth of 7 to 9 ply. This setting is close to tournament conditions, since in a tournament the thinking time is limited to five seconds per move. 100 starting positions were used to prevent that games were correlated. Each position was played with both colors, to remove the advantage of the initiative. 6,000 games were played to reach statistical significance. Due to the random factor in the evaluation function, different games were played even with the same initial position. The results are shown in Table 5.5. A 95% confidence bound is applied to the win ratio.

Table 5.5: Stratego selfplay experiment, 1 second per move.

t	ChanceProbCut	Regular	Win Ratio
12.8	2,981	3,019	49.7% ± 1.3%
6.4	3,009	2,991	50.1% ± 1.3%
3.2	3,102	2,898	51.7% ± 1.3%
2.4	3,043	2,957	50.7% ± 1.3%
2.0	3,002	2,998	50.0% ± 1.3%
1.8	3,108	2,892	51.8% ± 1.3%
1.6	3,072	2,928	51.2% ± 1.3%
1.4	3,059	2,941	50.9% ± 1.3%
1.2	3,066	2,934	51.1% ± 1.3%
1.0	3,107	2,893	51.8% ± 1.3%
0.8	3,113	2,887	51.9% ± 1.3%
0.6	3,035	2,965	50.6% ± 1.3%
0.5	3,073	2,927	51.2% ± 1.3%
0.4	3,084	2,916	51.4% ± 1.3%
0.3	3,077	2.923	51.3% ± 1.3%
0.2	3,114	2,886	51.9% ± 1.3%
0.1	3,063	2,937	51.1% ± 1.3%
0.05	3,018	2,982	50.3% ± 1.3%

For t values 0.2 and 0.8, a win ratio of 51.9% is achieved. For most other values of t, the program still performs well, taking into account the variance of the results. For t value 0.05, the search has become too selective and makes too many mistakes. For t values 6.4 and 12.8, not enough prunings are achieved to justify the overhead.

5.6.2 Dice

This subsection presents all the results obtained in the domain of Dice.

Determining Parameters

Because Dice has a regular game tree, chance nodes are counted as plies. We limit the applicability to $d \in \{7, 9\}$. R is set to 4 to handle the odd-even effect. On a test set of 1,000 5×5 positions, value pairs (v_{d-R}, v_d) have been determined and a regression line is calculated. We have chosen the 5×5 board for reference, because

Hauk (2004) has used this variant to test node reductions of the Star1 and Star2 techniques. In Figure 5.11 the regression model is shown for depths 3 and 7. Figure 5.12 shows the linear regression model for depths 5 and 9. These figures show that the linear regression model is suitable for estimating v_d.

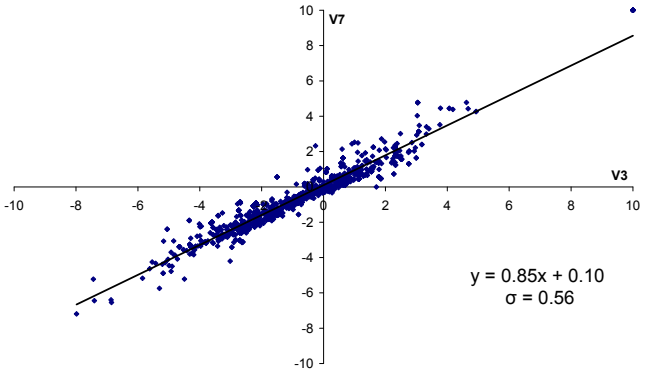


Figure 5.11: Evaluation pairs at depths 3 and 7 in Dice.

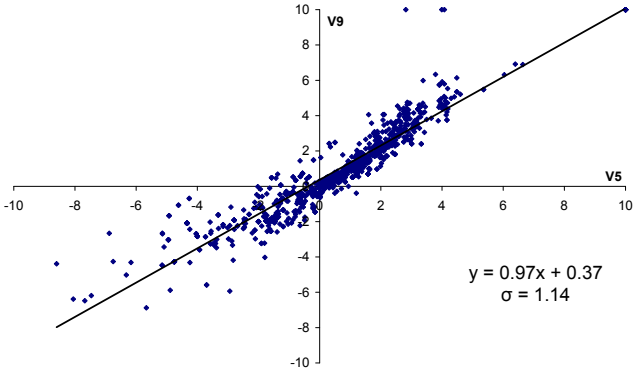


Figure 5.12: Evaluation pairs at depths 5 and 9 in Dice.

Tuning Selectiveness

Again, we have to find the optimal value for t. This tuning will be done in a similar fashion as described for Stratego. For this experiment, at depths 7 and 9 the regression model from Figure 5.11 and 5.12 are used and the t is varied. 1,000 positions were tested with 5 up to 12 checkers on the board. Tables 5.6, 5.7, and 5.8 give the results of tuning the t parameter for depths 9, 11 and 13, respectively.

In the three tables we observe that the average difference of the returned evaluation values increases when the t is decreased. Also the standard deviation of the

Table 5.6: Performance of ChanceProbCut at depth 9 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	190,822,592	-	-	-
3.2	163,926,279	14.1%	0.00	0.02
1.6	138,491,950	27.4%	0.01	0.05
0.8	112,607,261	41.0%	0.05	0.09
0.4	90,214,153	52.7%	0.10	0.15
0.2	72,684,352	61.9%	0.14	0.14
0.1	63,677,531	66.6%	0.16	0.15
0.05	59,189,953	69.0%	0.18	0.16

Table 5.7: Performance of ChanceProbCut at depth 11 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	2,322,871,121	-	-	-
3.2	1,945,429,167	16.2%	0.01	0.04
1.6	1,614,635,629	30.5%	0.02	0.05
0.8	1,292,924,069	44.3%	0.05	0.09
0.4	1,019,886,503	56.1%	0.09	0.11
0.2	830,281,706	64.3%	0.12	0.19
0.1	729,552,129	68.6%	0.14	0.15
0.05	679,098,869	70.8%	0.15	0.15

results grows when t is decreased. Table 5.6 shows that when using t=0.05 for depth 9 a reduction of 69.0% is achieved without a great loss in quality. Furthermore, we see that the majority of the game tree can be pruned without a large deterioration of quality. Finally, Table 5.8 shows that even a large reduction is obtained at search depth 13. Applying ChanceProbCut at a larger search depth did not lead to a reduction in quality.

Table 5.8: Performance of ChanceProbCut at depth 13 for 1,000 Dice positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	27,455,156,557	-	-	-
3.2	22,855,978,846	16.6%	0.01	0.02
1.6	18,819,031,785	31.5%	0.03	0.08
0.8	14,906,932,474	45.7%	0.06	0.11
0.4	11,707,480,207	57.4%	0.10	0.13
0.2	9,357,032,870	65.9%	0.13	0.14
0.1	8,313,736,712	69.7%	0.14	0.15
0.05	7,839,082,021	71.4%	0.15	0.15

A general observation for these tables is that the deeper the search, the larger the game-tree reduction is for the same t. In general, the search tree can be reduced by 50% before the ChanceProbCut value is mistaken by more than 0.10 point.

Selfplay

We decided to test ChanceProbCut on the 11×11 board. There are two reasons why a large board size has to be chosen. (1) Previous experiments in Dice were conducted on the 11×11 board (Hauk, 2004; Hauk *et al.*, 2006a). (2) With games such as Dice, it is easy to perform a deep search. Our engine is able to evaluate more than 2 million nodes per second. In non-deterministic games, an increase in search depth has limited influence on the playing strength after the first few plies. Due to these reasons, a large board has to be chosen to create an interesting variant.

Table 5.9 gives the results of the selfplay experiments of 20,000 games on the 11×11 board using 100 ms per move. With these time settings, the search engine reaches 9 plies in the opening phase, and 13 plies in the endgame. All games started from an empty board. Due to the dice element of the game and the random factor in the evaluation function, there exist no correlation between the games.

Table 5.9: 11×11 Dice selfplay experiment, 100 ms per move.

t	ChanceProbCut	Regular	Win Ratio
3.2	10,103	9,897	50.5% ± 0.7%
1.6	20,226	19,774	50.6% ± 0.5%
0.8	9,964	10,036	49.8% ± 0.7%
0.4	10,060	9,940	50.3% ± 0.7%
0.2	9,981	10,019	49.9% ± 0.7%
0.1	9,982	10,018	49.9% ± 0.7%
0.05	9,882	10,118	49.4% ± 0.7%

We see that ChanceProbCut does have a rather small, but genuine improvement in playing strength. With t set to 1.6, a win ratio of 50.6% ± 0.5% is achieved on 40,000 games.

5.6.3 ChanceBreakthrough

This subsection presents all the results obtained in the domain of ChanceBreakthrough.

Determining Parameters

In ChanceBreakthrough we do not count chance nodes as a ply. We limit the applicability to $d \in \{3, 5\}$. R is set to 2 to handle the odd-even effect. On a test set of 500 positions, generated with selfplay, value pairs (v_{d-R}, v_d) have been determined and a regression line is calculated. In Figure 5.13 the model is shown for depths 1 and 3. Figure 5.14 shows the linear regression model for depths 2 and 4. Finally,

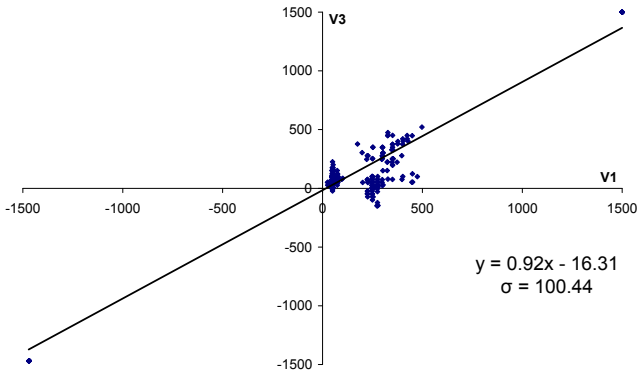


Figure 5.13: Evaluation pairs at depths 1 and 3 in ChanceBreakthrough.

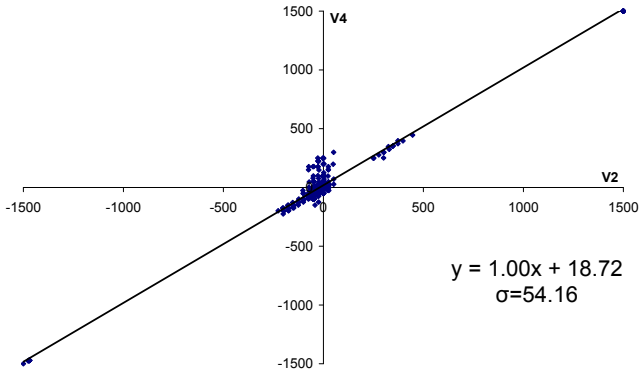


Figure 5.14: Evaluation pairs at depths 2 and 4 in ChanceBreakthrough.

Figure 5.15 depicts the model for depths 3 and 5. These figures show that linear regression is suitable for estimating v_d. It is harder to predict a 3-ply value based on a 1-ply search. However, the model stabilizes for deeper value pairs.

Tuning Selectiveness

We also tune t for the game of ChanceBreakthrough. The results for depth 5 on the test set of 500 positions can be seen in Table 5.10.

We see that a quite large error is being made. In a 5-ply search with $t = 3.2$, an error of 30.26 points is made on average (i.e., one seventh of a piece). Usually, the ChanceProbCut move does not make any error, but in some cases the returned value is far from correct, which is visible in the large standard deviation. These mistakes are always fatal in ChanceBreakthrough. Selfplay experiments should be concluded to find the appropriate t value.

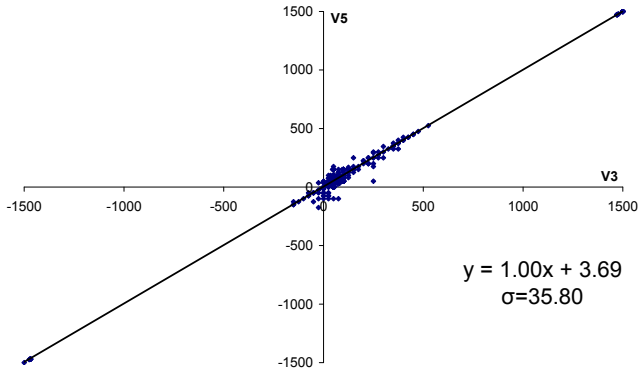


Figure 5.15: Evaluation pairs at depths 3 and 5 in ChanceBreakthrough.

Table 5.10: Performance of ChanceProbCut at depth 5 for 500 ChanceBreakthrough positions.

t	Nodes	Reduction	ACPCME	DCPCME
-	23,496,940,171	-	-	-
3.2	11,377,214,094	51.6%	30.26	172.66
1.6	10,826,472,281	53.9%	32.02	182.73
0.8	9,102,499,638	61.3%	33.17	187.83
0.4	7,463,883,129	68.2%	34.05	190.26
0.2	6,466,057,217	72.5%	35.12	191.43
0.1	5,637,776,939	76.0%	35.58	192.07
0.05	5,359,416,856	77.2%	35.80	192.30

Selfplay

Table 5.11 gives the results of the selfplay experiments of 2,000 games using 5 seconds per move. With these time settings, the search engine reaches around 5 plies. We have chosen to test a different range of t. Based on our experience with the engine, we believe that the optimal value of t was somewhere between 0.8 and 2.0. Due to the chance element of the game and the random factor in the evaluation function, there exists no correlation between the games.

In the table we see that ChanceProbCut is able to improve the playing strength significantly. For the best setting, a win ratio of 54.8% $\pm$ 2.2% was achieved, despite the occasional error, as discussed in the previous section. This is a larger gain than observed for Stratego and Dice. To be sure that this number is accurate, we played an additional 2,000 games for $t = 1.6$. For the total of 4,000 games, this setting was able to achieve a win ratio of 54.4% $\pm$ 1.5%.

Table 5.11: ChanceBreakthrough selfplay experiment, 5 seconds per move.

t	ChanceProbCut	Regular	Win Ratio
2.0	1,057	943	52.9% ± 2.2%
1.8	1,041	959	52.1% ± 2.2%
1.6	1,096	904	54.8% ± 2.2%
1.4	1,064	936	53.2% ± 2.2%
1.2	1,051	949	52.6% ± 2.2%
0.8	1,029	971	51.5% ± 2.2%

5.7 Chapter Conclusions and Future Research

In this chapter we have proposed the forward-pruning technique ChanceProbCut for expectimax. This technique is the first in its kind to forward prune at chance nodes.

ChanceProbCut is able to reduce the size of the game tree significantly without a loss of decision quality in Stratego, Dice, and ChanceBreakthrough. At depth 11 in Stratego, a safe reduction of 85.6% in the number of nodes is observed for t value 0.8. In Dice, a safe reduction of 31.5% of the game tree with 13 plies can be achieved, using t value 1.6. At depth 5 in ChanceBreakthrough, ChanceProbCut prunes 53.9% of the search tree for $t = 1.6$. Thus, the first conclusion we may draw, is that ChanceProbCut finds a good move faster in the expectimax framework, while not affecting the playing strength. Because ChanceProbCut finds a good move faster, one might consider different approaches of investing the gained time. For instance, this time can be utilized for a more time-consuming evaluation function.

Selfplay experiments in Stratego and Dice reveal that there is a small improvement in playing strength, which is still relevant. In Stratego, ChanceProbCut achieves a win ratio of 51.9% ± 1.3% and in Dice 50.6% ± 0.5%. The rather small increase in playing strength is due to the nature of expectimax. We point out two reasons. (1) The result of a game is dependent on luck. Even a weak player may win some games. (2) Deeper search has a small influence on the playing strength of expectimax, compared to minimax. For Dice, Hauk *et al.* (2006b) showed that searching 9 plies instead of 5 increased the win ratio by only 2.5%. A similar phenomenon was observed in Backgammon. If we take this into account, ChanceProbCut performs rather well. In ChanceBreakthrough however, a significant increase in performance was measured. ChanceProbCut was able to win 54.4% ± 1.5% on 4,000 games. The second conclusion we may draw, is that ChanceProbCut improves the playing strength.

We propose four directions for future research. (1) For improving the effectiveness of ChanceProbCut, more value pairs may be used. (2) The regression parameters and cut-threshold t can be bootstrapped according to the game phase. (3) A successor of ProbCut exists, called Multi-ProbCut (Buro, 2000). This technique could also be adapted for chance nodes (Multi-ChanceProbCut). (4) ChanceProbCut should be applied to other non-deterministic or imperfect-information games to test its effectiveness.

Chapter 6

Best-Reply Search in Multi-Player Games

This chapter is an updated and abridged version of the following publication:

1. Schadd, M.P.D. and Winands, M.H.M. (2011). Best Reply Search for Multiplayer Games. *Transactions on Computational Intelligence and AI in Games*, Vol. 3, No. 1, pp. 57–66.

In deterministic two-player games with perfect information, the majority of research has focused on the minimax algorithm (Knuth and Moore, 1975). For deterministic perfect-information multi-player games, the choice of search algorithm is not as straightforward. The reason is that with multiple players a (temporary) coalition may be formed. The influence of coalitions separates multi-player games into cooperative and non-cooperative games. For the former, a player is able to achieve its maximum score only if that player forms a coalition with other players. For the latter, a player is able to achieve its maximum score if it plays individually. However, the other players may prevent that a player achieves its maximum score by forming a (temporary) coalition. In this chapter, we focus on non-cooperative multi-player games.

The two main search algorithms for non-cooperative multi-player games are $\max^n$ (Luckhardt and Irani, 1986) and paranoid (Sturtevant and Korf, 2000), each approaching this class of games from a different angle. Max^n assumes that every player tries to maximize the own score, while paranoid assumes that all opponents form a coalition against the root player. Max^n and the paranoid algorithm may have conceptual drawbacks. Due to the lack of pruning in $\max^n$ only a limited lookahead is possible (Sturtevant, 2003b). Furthermore, the underlying assumption of $\max^n$ may be unrealistic, resulting in $\max^n$ to be too optimistic (Zuckerman *et al.*, 2009). Just changing the tie-breaking rule for the $\max^n$ algorithm can have arbitrary results for the value of the root (Sturtevant, 2003c). When performing a deep search with the paranoid algorithm, the other players may dominate the root player (Saito and Winands, 2010), resulting in paranoid to be too pessimistic.

This chapter answers the fourth research question by proposing Best-Reply Search (BRS). This search algorithm tries to overcome the problems of the max^n and paranoid algorithms. For multi-player games, we assume that not every opponent is trying to minimize the root player's score. Instead, only one opponent is minimizing the root player's score. BRS chooses which opponent is allowed to play a counter move at a so-called MIN node in the search tree. The selected opponent is the one that has the strongest counter move against the root player. The other players have to pass their turn. By searching in this way, a significant lookahead can be achieved even with many opponents. Furthermore, the playing style is less cautious compared to the paranoid algorithm. We apply BRS in three domains, Chinese Checkers, Focus and Rolit.

The chapter is organized as follows. First, we discuss the difference between cooperative and non-cooperative games in Section 6.1. Next, the max^n and the paranoid algorithms are presented in Section 6.2. Subsequently, BRS is introduced in Section 6.3. Thereafter, Section 6.4 describes the test domains, the games Chinese Checkers, Focus and Rolit. Section 6.5 presents the experiments. Finally, Section 6.6 gives the conclusions and an outlook on future research.

6.1 Coalition Forming in Multi-Player Games

A multi-player game may be classified either as cooperative or non-cooperative. A game is called a *cooperative game*[1] if it consists of a number of players N and a function v that associates with every nonempty subset S of N (a coalition) a real number $v(S)$ (i.e., the *worth* of S) (Osborne and Rubinstein, 1994). In a cooperative game, there typically exist coalitions of which the payoff of a coalition is higher than for each individual. There are three aspects in which *non-cooperative games* differ to cooperative games. (1) At the end of the game, $v(S) = \frac{1}{|S|}$, implying that winning solely is preferred above drawing the game. (2) During the game, the value of $v(S)$ may change, implying that coalitions may be of temporary nature. We define $v(S,t)$ as the value of coalition S at time step t. For example, if a player N_i is ahead, then the value of its private coalition $v(\{N_i\}) = 1$ and the value of each other player's private coalition is worth 0. The set of losing players $L = N \backslash \{N_i\}$ may have $v(L,t) > 0$ because they are able to catch up with the winning player if they cooperate. (3) The value of $v(L,t)$ is not globally known. Every player may have hidden preferences, based on their own evaluation function and used search algorithm. These preferences are only indicated by making moves on the board. Therefore, non-cooperative multi-player games are an ideal testbed for dealing with temporary coalition forming with unknown preferences.

Modal logics have been used to develop formal models for reasoning about coalitions in cooperative games. These models address three aspects of coalitions, mostly separately (cf. Kurzen, 2009). (1) Which results can coalitions achieve? (Pauly, 2002) (2) How can they achieve something? (Harel, 1984) (3) Why would they want to achieve a certain result? (Van Benthem, Girard, and Roy, 2008) There exist efforts to combine these models (1+2, Sauro *et al.*, 2006; 1+2+3, Kurzen, 2009). It is

[1] Also called a *cooperation* or *coalition* game.

an open question how modal logics may be adapted to be applicable to a search algorithm for non-cooperative multi-player games. Generally, existing search algorithms for non-cooperative games, such as the paranoid algorithm (Sturtevant and Korf, 2000), assume that coalitions are static and do not change during the search and often even during the complete game. Two exceptions are the comixer algorithm (Lorenz and Tscheuschner, 2006) and the MP-Mix algorithm (Zuckerman *et al.*, 2009). Finally, we remark that Peterson, Reif, and Azhar (2002) proposed an algorithm that uses Turing machines to decide whether a coalition is able to win various types of multi-player games of imperfect information.

6.2 Search Algorithms for Multi-Player Games

This section discusses two well-understood search algorithms for deterministic multi-player turn-taking games with perfect information, the max^n (Luckhardt and Irani, 1986) and the paranoid algorithm (Sturtevant and Korf, 2000). We first introduce the max^n algorithm in Subsection 6.2.1 and thereafter discuss the paranoid algorithm in Subsection 6.2.2.

6.2.1 Max^n

The max^n algorithm (Luckhardt and Irani, 1986) can be used in games with any number of players. At a leaf node, an n-tuple is generated, where n is the number of players and every entry corresponds to the score a player receives. At internal nodes, a player chooses the child with the highest score for that player. An example max^n tree is depicted in Figure 6.1.

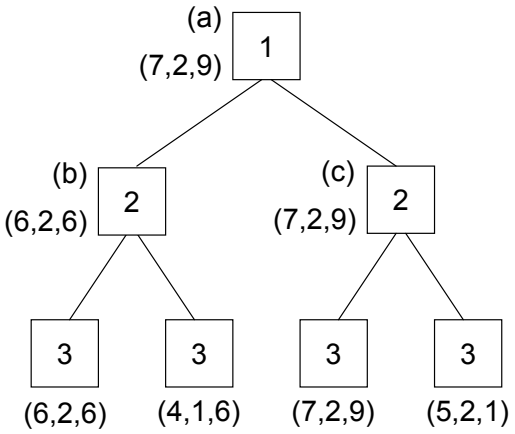


Figure 6.1: An example max^n tree.

Nodes (a), (b) and (c) are internal nodes. In Node (a), Player 1 is to move. In Nodes (b) and (c), Player 2 is to move. Player 2 has the choice between (6,2,6) and

(4,1,6) in Node (b) and chooses (6,2,6) to maximize the own score. In Node (c), both options are equally good for Player 2. In this example the left child is chosen and Node (c) gets value (7,2,9). Player 1 now prefers Node (c) above Node (b), because it gives a higher score for Player 1.

In multi-player games, there may exist multiple equilibrium points. It has been proven that $\max^n$ computes one equilibrium point (Luckhardt and Irani, 1986). Furthermore, if the tie-breaking rule is altered, it may arbitrarily affect the $\max^n$ value of the tree (Sturtevant, 2003c). We demonstrate this by changing the tie-breaking rule in Figure 6.1. Instead of choosing the left-most child in case of a tie, now the policy is used that the child which has the lowest value for the root player is chosen. In this case, Node (c) has value (5,2,1). This implies that the Player 1 prefers Node (b) over (c) and Node (a) has value (6,2,6) instead of (7,2,9).

The weakness of $\max^n$ is twofold. (1) Due to the lack of safe pruning only a limited lookahead is possible (Sturtevant, 2003a). Safe pruning is possible in $\max^n$ under certain conditions. One of these conditions is that there is an upper bound on the sum of scores of all players (i.e., there is a certain number of points to be distributed over all players). In this case, shallow pruning is possible (Korf, 1991). Deep pruning is, however, not possible. Sturtevant (2003b) shows that less-shallow pruning is possible, up to n ply deep. A pruning may occur when intermediate players between the first and last player are all on their last branch of their search (last-branch pruning) or earlier when re-searching certain branches (speculative pruning). (2) The second weakness is that the underlying assumption of $\max^n$ may be unrealistic. The $\max^n$ algorithm assumes no coalition forming of the opponents. The result may be that $\max^n$ is too optimistic. To make $\max^n$ somewhat more cautious, the tie-breaking rule which assumes the worst case for the root player is used in this chapter. To further increase the cautiousness of $\max^n$, a "paranoid" evaluation function (i.e., it assumes that all opponents have formed a coalition) may be considered. Several other variations of the $\max^n$ algorithm exist, which try to overcome both weaknesses. Careful $\max^n$ (Lorenz and Tscheuschner, 2006) uses a weighted-average update rule to model uncertainty if the opponent has multiple good moves. The comixer algorithm (Lorenz and Tscheuschner, 2006) considers possible coalitions against the strongest player at every node of the search tree. This may be a correct assumption if a player is ahead, but might lead to weak play if no such coalition exists. For handling imperfect opponent models, two variations have been introduced, soft-$\max^n$ (Sturtevant and Bowling, 2006) and prob-$\max^n$ (Sturtevant, Zinkevich, and Bowling, 2006).

6.2.2 Paranoid

The paranoid algorithm (Sturtevant and Korf, 2000) reduces the multi-player game to a two-player game by making the "paranoid" assumption. The algorithm assumes that all opponents have formed a coalition against the root player. By doing so, regular $\alpha\beta$ pruning is possible. This leads to a larger search depth. Figure 6.2 depicts an example of the paranoid algorithm. It is the same tree as in Figure 6.1, but now the leaf nodes are evaluated in a paranoid way. Here, the sum of the evaluation scores of Player 2 and 3 are subtracted from the evaluation score of

Player 1 (Sturtevant, 2003a). A second possibility is that Players 2 and 3 ignore their own score, and only minimize Player 1's score (Sturtevant, 2003c). In this chapter we apply the first approach. In Node (b), the right child is preferred with value -3. After finding -4 as value of the first child of Node (c), all the remaining children can be safely pruned according to the standard $\alpha\beta$ rule. The root node (a) receives value -3.

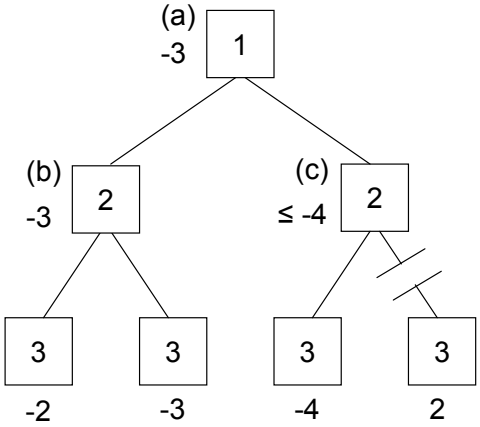


Figure 6.2: An example paranoid tree.

In the best case, $O(b^{d/2})$ nodes are expanded for two-player games (Knuth and Moore, 1975), where b is the average branching factor and d the search depth. Sturtevant (2003c) showed that the paranoid algorithm expands $O\left(b^{d\times(n-1)/n}\right)$ nodes in the best case for multi-player games, which is a generalization of the best case for two-player games. Paranoid may outperform $\max^n$ due to the larger lookahead (e.g., for Chinese Checkers or Hearts) (Sturtevant, 2003a).

Due to the unrealistic paranoid assumption, it is possible that suboptimal play occurs (Sturtevant and Korf, 2000). Furthermore, if an infinite amount of time would be available, the root player might assume that all moves are losing, leading to poor play. For the game of Rolit, Saito and Winands (2010) showed that for three players on the 6×6 board, the first and second player cannot gain any points under the paranoid assumption (and the third player only 1 point because he places the last stone). Usually, it is not possible to win when all opponents form a coalition. As a general observation, the deeper the search goes, the more pessimistic the value of the root becomes.

6.3 Best-Reply Search

Sturtevant (2003a) proposed a table-based evaluation function for Chinese Checkers which assumes solitary play. Although this assumption is unrealistic, this evaluation function proved to work well. Inspired by this result, we investigate in this section

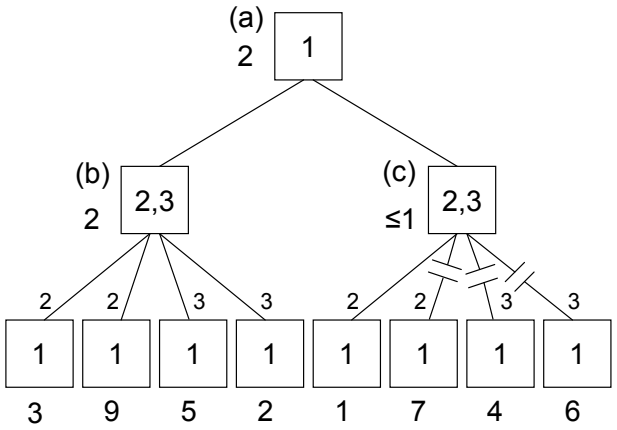


Figure 6.3: An example BRS tree.

whether this idea may be transferred from the evaluation function to the search tree. We propose Best-Reply Search (BRS) to overcome some weaknesses of the $\max^n$ and paranoid algorithms.

Subsection 6.3.1 presents the underlying idea behind BRS. The pseudo code of BRS is given in Subsection 6.3.2. In Subsection 6.3.3 the best-case analysis of BRS is given. Finally, Subsection 6.3.4 discusses strengths and weaknesses of BRS.

6.3.1 Idea

Traditional search methods allow every player to make a move, resulting in large search trees. In BRS, not every opponent is allowed to make a move. Only the opponent with the strongest move against the root player may move. At a MIN node, all moves for all opponents are searched. It means that an opponent is allowed to make a move even if it is not its turn. At the following MAX node, it is again the root player's turn. BRS achieves long-term planning because more MAX nodes are visited along the search path while being at the same time less paranoid. When iterative deepening is applied in BRS, a MIN node always contains all players. When comparing this to the standard search depth of the paranoid algorithm, the search depth is increased irregularly. For instance with 4 players, the search depth of BRS is increased to depth 1, 4, 5, 8, 9, etc., with respect to the standard depth. An example of BRS is depicted in Figure 6.3.

Nodes (b) and (c) are labeled '2,3', which represents that one of the opponents (i.e., Player 2 or 3) is allowed to make a counter move. The labels near the edges underneath Nodes (b) and (c) indicate which player's move is played. At the next ply, it is again Player 1's turn to play. Node (b) is assigned value 2, because Player 3 has the strongest counter move. When searching the first child of Node (c), which is a move by Player 2, a regular $\alpha\beta$ pruning occurs. In this case, the remaining moves of Player 2 and all moves of Player 3 are pruned.

```
 1: BRS(alpha, beta, depth, turn)
 2:
 3: if depth ≤ 0 then
 4:   return eval()
 5: end if
 6:
 7: if turn == MAX then
 8:   Moves = GenerateMoves(MaxPlayer);
 9:   turn = MIN;
10: else
11:   for all Opponents o do
12:     Moves += GenerateMoves(o);
13:   end for
14:   turn = MAX;
15: end if
16:
17: for all Moves m do
18:   doMove(m);
19:   v = −BRS(−beta, −alpha, depth−1, turn);
20:   undoMove(m);
21:
22:   if v ≥ beta then
23:     return v;
24:   end if
25:   alpha = max(alpha, v);
26: end for
27:
28: return alpha;
```

Algorithm 6.1: Best-Reply Search.

6.3.2 Pseudo Code

Algorithm 6.1 shows the pseudo code for BRS. The first change to the standard $\alpha\beta$ algorithm (for the negamax framework) is shown in lines 7−15. If the current node is a MAX node, the moves are generated as usual (Line 8). If it is a MIN node the moves for all opponents are generated (Lines 11−13). Before performing the traversal of all moves (Line 17), the type of node should have been altered at every ply (Lines 9 and 14).

6.3.3 Best-Case Analysis of BRS

If pruning is not feasible, $\max^n$ has to examine the complete search tree. With average branching factor b and search depth d, $\max^n$ searches $O\left(b^d\right)$ nodes. Sturtevant (2003c) showed that the paranoid algorithm explores $O\left(b^{d\times(n-1)/n}\right)$ nodes in the best case, where n is the number of players. Analogous to their proof, we can prove the best case of BRS.

Theorem. Best-Reply Search explores in the best case $O\left((b \times (n-1))^{\lceil \frac{2 \times d}{n} \rceil / 2}\right)$ nodes.

Proof. Assume a uniform tree is searched until depth d. In BRS, this search depth is reduced to $\lceil \frac{2 \times d}{n} \rceil$ because the layers of n successive players is reduced to 2 layers. The branching factor b is increased to $b \times (n-1)$ at MIN nodes, assuming that the opponent moves do not interact with each other. To calculate the minimum number of nodes that have to be examined within the game tree, we need a strategy for the MAX and MIN player. For finding a strategy for the MAX player, 1 move has to be searched at a MAX node, and $b \times (n-1)$ moves at a MIN node, resulting in $(b \times (n-1))^{\lceil \frac{2 \times d}{n} \rceil / 2}$ nodes. For finding a strategy for the MIN player, the collection of all opponents, 1 move has to be searched at a MIN node and b moves at a MAX node, resulting in $b^{\lceil \frac{2 \times d}{n} \rceil / 2}$ nodes. Therefore, the total number of nodes by both the MAX and MIN player is $(b \times (n-1))^{\lceil \frac{2 \times d}{n} \rceil / 2} + b^{\lceil \frac{2 \times d}{n} \rceil / 2}$ nodes. Thus, BRS explores in the best case $O\left((b \times (n-1))^{\lceil \frac{2 \times d}{n} \rceil / 2}\right)$ nodes. □

We remark that for two players, the best case of BRS is identical to the best case of $\alpha\beta$, which is $O(b^{d/2})$.

6.3.4 Strengths and Weaknesses of BRS

We point out two advantages of BRS over the max^n and paranoid algorithms. (1) More MAX nodes are visited along the search path, leading to more long-term planning. (2) It softens the unrealistic max^n and paranoid assumptions. Max^n assumes that there are no coalitions, while paranoid assumes that all opponents form a coalition against the root player. An additional advantage over max^n is that BRS may be able to prune parts of the tree.

We point out two drawbacks of BRS as well. (1) Not all players are allowed to make a move, leading to illegal positions. (2) Opponent moves which are beneficial for the root player might not be considered.

For trick-based card games, such as Hearts and Spades, BRS is not an appropriate method. The first problem is that the first player to play a card in a trick determines which suit is played. If BRS would be applied here, a suit may be played which the first player does not have, creating an illegal position that is considerably different from a legal position. The second problem occurs when not all opponents play a card during a trick. This causes that players have a different number of cards in their hands, and it is not defined what happens at the end of the game. A third problem in the game of Hearts is that playing the Ace of Hearts card may result in only gaining 2 points instead of 4 when applying BRS. Problems such as these make BRS not applicable to trick-based card games.

6.4 Test Domain

To test whether BRS works well, we use three non-cooperative deterministic multi-player games with perfect information, Chinese Checkers, Focus, and Rolit. Chinese

Checkers is a race game and the rules are given in Subsection 6.4.1. Focus is a material-based game and is explained in Subsection 6.4.2. In Subsection 6.4.3 the rules of the territorial-based game Rolit are given. Rolit is the multi-player version of Othello.[2] The game engines are described at the end of each subsection.

6.4.1 Chinese Checkers

Chinese Checkers is a board game that can be played by two to six players. It was invented in 1893 and has since then been released by various publishers under different names. Chinese Checkers is played on a star-shaped board. The most common board contains 121 fields, where each player starts with 10 pieces. We decided to play on a slightly smaller board (Sturtevant, 2008a) (see Figure 6.4). In this version, each player plays with 6 pieces. The advantage of a smaller board is that it allows us to use a strong evaluation function (Sturtevant, 2003a).

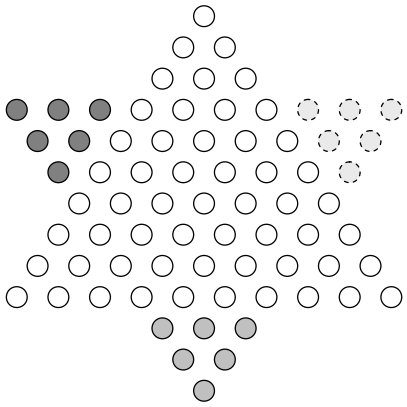


Figure 6.4: A three-player Chinese Checkers board.

The goal of each player is to move the own pieces to the own base at the other side of the board. Pieces may move to one of the adjacent squares or they may jump over another piece to an empty field. A player may also make multiple jumps with one piece in one turn. It is possible to create a setup that allows pieces to jump over a large distance. The first player who manages to fill the home base wins the game. To avoid blocking behavior, the player wins the game when the home base is filled *and* the player owns at least one of the pieces in the home base.

Engine

To evaluate a board position, we use a lookup table which stores the number of moves a single player would require to finish the game (Sturtevant, 2008b). This number does not take into account opponent pieces, which results in erroneous evaluation in the middle game. In the endgame, this lookup table allows perfect play. The value

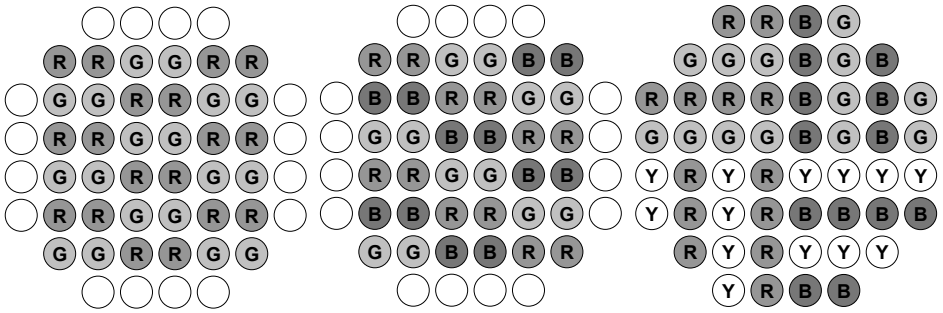
[2] Also known as Reversi.

found in the table was multiplied by 1,000. Additionally, a random factor of 5 points was included to prevent games from being repeated (due to deterministic play) in the experiments.

The moves are ordered statically such that moves which approach the home base the most are investigated first (e.g., long jumping moves towards the home base).

6.4.2 Focus

Focus is an abstract multi-player strategy board game, invented in 1963 by Sid Sackson (Sackson, 1969). This game has also been released under the name Domination. Focus is played on an 8×8 board where in each corner 3 squares are removed. It can be played by two, three or four players. Each player starts with a number of pieces on the board. In Figure 6.5, the initial board positions for the two-, three- and four-player variants are given. The letters R, G, B, and Y correspond to the piece colors of the game: red, green, blue, and yellow, respectively.



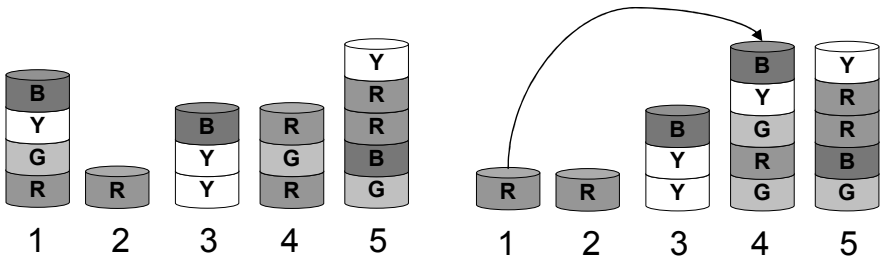
(a) Two players (b) Three players (c) Four players

Figure 6.5: Setups for Focus.

In contrast to many other games, pieces in Focus may be stacked on top of each other. Each turn a player may move a stack, which contains one or more pieces, orthogonally as many squares as the stack is tall. The player may only move a stack of pieces if a piece of their color is on top of the stack. Players are also allowed to split stacks in two smaller stacks. If they decide to do so, then they only move the upper stack as many squares as the number of pieces in that stack.

If a stack lands on another stack, then the stacks are merged. If the merged stack has a size of $n{>}5$, then the bottom $n{-}5$ pieces are captured by the player, such that there are 5 pieces left. If a player captures one of the own pieces, the player may later choose to place an own piece back on the board, instead of moving a stack.

An example move is depicted in Figure 6.6. Here, Blue chooses to move three pieces of Stack 1, three positions to the right. By this move, the control of Stack 1 is transferred to Red, which owns the highest piece of Stack 1 after Blue has moved. Stack 4 would contain 6 pieces after Blue has moved, indicating that a capture shall take place. Only the bottom red piece is captured because 5 pieces are allowed per stack.



(a) Blue to move. (b) Blue has captured a piece of Red.

Figure 6.6: Example move for Focus.

There exist two variations of the game, each with a different winning condition. In the standard version of the game, a player has won if all other players cannot perform a legal move. However, these games may take a long time to finish. Therefore, in this chapter the shortened version of the game is used. In this version, a player has won if either a certain number of pieces or a certain number of pieces from each opponent have been captured. In the two-player variant, a player wins if at least 6 pieces from the opponent are captured. In the three-player variant, a player has won if either at least 10 pieces in total or at least 3 pieces from each opponent are captured. Finally, in the four-player variant, the goal is to either capture 10 opponent pieces or at least 2 pieces from each opponent. If the game is not decided after 300 moves, it is scored as a draw.

Engine

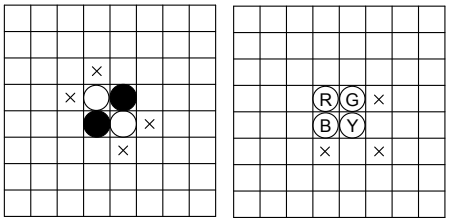
The evaluation function consists of two parts. (1) The first term is the minimum number of pieces needed to finish the game for either finish condition. This number is multiplied by 1,000. (2) The second term is the position of pieces in a stack. The higher the piece on a stack of pieces, the more points it is worth. Being on top of a stack gives control of the stack, and this is especially valuable if the stack is tall. Furthermore, the higher a piece, the more difficult it is to capture it. For every own piece its height is squared and added to the total score. A small random factor of 5 points is included to prevent repetition.

The static move ordering consists of two parts. (1) Moves which involve a large number of pieces (pieces of the moved stack plus pieces of the target stack). (2) Moves which increase the number of stacks a player controls. The first term is the dominant one.

6.4.3 Rolit

Rolit is a multi-player variant of the well-known game Othello. Therefore, we start with a description of this game before turning to Rolit. Othello is a deterministic two-player game with perfect information played on an 8×8 board. The players are called Black and White and their objective is to maximize their number of pieces

on the board. The initial position is shown in Subfigure 6.7(a). A player may only place a piece as part of a flipping move and a player has to pass if no such move is available. A *flipping move* places a piece at the end of a flipping line. A *flipping line* is a straight line of squares on the board such that four conditions are met: (1) The squares on the line have to be all vertically, all horizontally, or all diagonally connected. (2) All squares on the line are occupied. (3) Both ends of the line must contain a piece of the player to move. (4) All pieces between are of the opponent. If a flipping move is played, the opponent pieces between the ends of the flipping line are flipped over to its own color.[3] If a move enables more than one flipping line, all flipping lines are executed. The game ends when the board is completely occupied or one player has no pieces left. The player with more pieces on the board wins the game.



(a) Othello (b) Rolit

Figure 6.7: Setups for Othello and Rolit. Legal moves are marked with ×.

Rolit is the multi-player generalization of Othello and has only minor differences in the rules. It may be played with two, three and four players, called Red, Green, Yellow and Blue. The initial position, which is identical for the two-, three- and four-player variant, is depicted in Subfigure 6.7(b). This implies that when playing with two or three players, there initially exist pieces on the board which do not belong to any of the players. Red starts the game and immediately has the opportunity to eliminate an opponent. In order to allow a player who has no piece left on the board to come back into the game, the rules are changed compared to Othello. When no flipping move is available (either because a player is eliminated or because there are just no flipping moves available) a player is allowed to place a piece on an empty field. The empty field has to be horizontally, vertically, or diagonally adjacent to a piece on the board. The game is scored analogously to Othello.

Engine

Our evaluation function is pattern based, inspired by the work of Buro (2003). Almost 100,000 games of the WTHOR database[4] were analyzed on 10 different patterns for 12 stages of the game. These patterns are the orthogonal lines of size 8 (4 lines due to symmetry), the entire diagonals of at least size 4 (5 diagonals due to

[3]The pieces are black one side, and white on the other.
[4]http://www.ffothello.org/info/base.php

symmetry) and a 2×4 corner region (Buro, 2003). These add up to a total number of 511,272 patterns. For each pattern the average score at the end of the game was computed. All the pattern scores are averaged for evaluating a position, and a random factor of 0.1 is added. These patterns are based on two players, while we need to evaluate positions for three and four players in Rolit. To bridge the gap, we assume that all opponent pieces have the same color when looking up pattern values in the database. It would be more accurate to create a pattern database for three and four players, but we abandoned this idea due to the combinatorial explosion and the unavailability of a Rolit database. The pattern-based approach, though, was superior to our original evaluation function that employed standard Othello features.

Our static move ordering prefers moves on squares which are known to be good. For example, the most preferred ones are the corners. The squares adjacent to the corners are the least preferred.

6.5 Experiments and Results

In this section, the experiments with BRS are presented for the games of Chinese Checkers, Focus, and Rolit. The performance of BRS is tested with three different time settings (250, 1000, 5000 ms) and with three, four and/or six players against one type of opponent ($\max^n$ or paranoid). In a three-player game there are $2^3 = 8$ different player-type assignments. Games where only one type of algorithm is playing are not interesting, leaving 6 ways to assign player types. For four players, there are $2^4 - 2 = 14$ assignments, and for six players, there are $2^6 - 2 = 62$ assignments (Sturtevant, 2003c). Each assignment is played multiple times until at least 1,000 games are reached and each assignment was played equally often. The random factor in each evaluation function prevented board repetition. All experiments were performed on an AMD64 2.4 GHz computer.

The following techniques were used if not mentioned otherwise. All algorithms used two-deep transposition tables (Greenblatt *et al.*, 1967; Breuker, 1998) and iterative deepening until the available time was depleted. Furthermore, paranoid and BRS used the history heuristic (Schaeffer, 1983) and killer moves (Akl and Newborn, 1977). In all columns labeled 'Win Ratio' a 95% confidence interval is applied.

Advanced pruning techniques were not used in all three algorithms. For $\max^n$, speculative pruning is able to significantly increase the playing strength (Sturtevant, 2003a; Sturtevant, 2003b). However, the tight upper bounds that are required are not present for our evaluation function. For the paranoid algorithm and BRS, well-known forward-pruning techniques could be used as well. These include the null-move heuristic (Beal, 1989; Goetsch and Campell, 1990), ProbCut (Buro, 1995) and Multi-Cut (Björnsson and Marsland, 2001). In order to successfully apply these forward-pruning techniques, the preconditions and parameters need to be tuned rather well. We did not enable any of these techniques because the effect of each of them is domain dependent.

In Subsection 6.5.1, we present the experiments to validate the implementation. The average search depth that the different algorithms are able to achieve is shown in

Subsection 6.5.2. Subsection 6.5.3 presents the performance of BRS against max^n. In Subsection 6.5.4, the experiments of BRS against paranoid are discussed. Subsection 6.5.5 shows experiments with three players, where there is 1 BRS player, 1 paranoid player and 1 max^n player.

6.5.1 Validation

To check whether the implementation of max^n and paranoid is similar to Sturtevant (2003c), we reconstructed his experiments. Paranoid played 600 three-player Chinese Checkers games against max^n using 250 ms (approximately 250k nodes). The results are presented in the lower part of Table 6.1. With the basic settings, max^n and paranoid did not use any additional techniques. In the advanced settings, max^n and paranoid used transposition tables, and paranoid was furthermore allowed to use the history heuristic and killer moves. The results of the basic setting confirm the results by Sturtevant (2003c). He reports a win ratio of 60.6% for his paranoid program. Out of curiosity, we also performed this experiment for the two-player version of Chinese Checkers. The results of 1,000 games are given in the upper part of Table 6.1. Here it is clear that paranoid, which is now a regular $\alpha\beta$ search, outperforms max^n due to larger lookahead. The reason for the advanced setting winning fewer games in the two-player variant might be that it plays more defensive and takes less risk.

Table 6.1: Winning statistics for Paranoid vs. Max^n for Chinese Checkers with 250 ms per move.

Two-player	Paranoid	Max^n	Win Ratio
Basic Settings	955	45	95.5% ± 1.3%
Advanced Settings	917	83	91.7% ± 1.7%
Three-player	**Paranoid**	**Max^n**	**Win Ratio**
Basic Settings	350	250	58.3% ± 3.9%
Advanced Settings	474	126	79.0% ± 3.3%

If BRS is applied to a two-player game, it should behave identical to the paranoid algorithm. For verification purposes, paranoid was matched against BRS for two-player Chinese Checkers, Focus and Rolit. There may be a slight overhead in BRS due to the move generation, but a win ratio of 50% should be expected. 1,000 games were played for each setting and game. Table 6.2 shows that for the two-player version of all three games, paranoid and BRS are equally strong.

6.5.2 Average Search Depth

The average search depth, which max^n, paranoid, and BRS can achieve in Chinese Checkers, Focus, and Rolit with different time settings and different number of players, is shown in Table 6.3.

Here we see that in all games paranoid is always able to search deeper than max^n. In Chinese Checkers, paranoid performs close to the best case, or even better (e.g.,

Table 6.2: Winning statistics for Paranoid vs. BRS for two-players with 250 ms per move.

Chinese Checkers	Paranoid	BRS	Win Ratio
Basic Settings	527	473	52.7% ± 3.1%
Advanced Settings	475	525	47.5% ± 3.1%
Focus	**Paranoid**	**BRS**	**Win Ratio**
Basic Settings	529	471	52.9% ± 3.1%
Advanced Settings	506	494	50.6% ± 3.1%
Rolit	**Paranoid**	**BRS**	**Win Ratio**
Basic Settings	476.5	523.5	47.7% ± 3.1%
Advanced Settings	487.5	512.5	48.8% ± 3.1%

the best-case depth for four players with 5 seconds thinking time is approximately $4.0 \times \frac{4}{3} \approx 5.3$). This performance is due to the effective move ordering. For instance, 1.1 moves are searched in CUT nodes (Marsland and Popowich, 1985) on average for three players, and only 1.05 for four players. Moreover, there are two explanations how it is possible to perform better than the theoretical best case, $O(b^{d(n-1)/n})$. (1) Because only complete plies are counted, the data is coarse-grained. (2) The paranoid algorithm can perform better than $O(b^{d(n-1)/n})$ if the domain-dependent move ordering prefers slim subtrees above large subtrees, taking advantage of the non-uniform nature of the game tree (i.e., a variable branching factor and search depth, cf. Plaat, 1996). For Chinese Checkers these are moves which enter the own goal area, because once entered, pieces are not allowed to leave anymore. In the games of Focus and Rolit, paranoid is close to the best case.

For all games BRS achieves a similar search depth as paranoid for every setting. However, these numbers are not strictly comparable due to the different search approaches. One conclusion we may draw is that BRS visits along the search path at least as many MAX nodes as paranoid. For example, for six players and 5 seconds thinking time, paranoid and BRS search approximately 5 ply. Due to the large number of players, paranoid only visits 1 MAX node, which is the root node. BRS is able to visit 3 MAX nodes in this case.

6.5.3 BRS against Max^n

Table 6.4 shows the winning performance of BRS against max^n for Chinese Checkers, Focus, and Rolit. In Chinese Checkers there are no draws possible. For Focus, draws are counted as $\frac{1}{3}$ or $\frac{1}{4}$ point for the three- and four-player variants, respectively. In Rolit, a draw may be shared between a subset of players. For example, if there are 2 BRS players and 1 paranoid player, both BRS players may share the highest score. The winning players receive the corresponding fraction of a point.

For Chinese Checkers we see that max^n is outperformed by BRS. In the worst case, a win ratio of 72.4% ± 2.8% is still achieved. In most cases, the performance is approximately 80%. In the best case, a win ratio of 88.0% ± 2.0% is achieved with 1,000 ms per move. We observe that in general, BRS gets stronger with more thinking time. In the material-based game Focus, we observe that BRS is outperforming

Table 6.3: Average search depth.

Chinese Checkers				
Players	Time (ms)	Max^n	Paranoid	BRS
3	250	3.0	4.9	4.7
3	1,000	3.2	5.0	5.0
3	5,000	4.0	5.7	6.0
4	250	3.0	4.8	4.5
4	1,000	3.6	5.1	5.0
4	5,000	4.0	5.9	5.5
6	250	3.0	3.9	3.9
6	1,000	3.5	4.2	4.9
6	5,000	4.0	4.9	5.0

Focus				
Players	Time (ms)	Max^n	Paranoid	BRS
3	250	3.0	4.5	4.6
3	1,000	3.2	5.0	5.0
3	5,000	3.9	5.5	5.8
4	250	3.0	4.1	4.4
4	1,000	3.3	4.9	5.1
4	5,000	4.0	5.5	5.6

Rolit				
Players	Time (ms)	Max^n	Paranoid	BRS
3	250	3.3	5.0	5.1
3	1,000	4.1	5.7	5.9
3	5,000	4.6	7.0	7.3
4	250	3.4	5.0	4.8
4	1,000	4.2	5.9	5.4
4	5,000	4.9	6.6	6.7

max^n easily. In the worst case, a win ratio of 81.2% $\pm$ 2.4% is still achieved. The max^n algorithm plays the strongest in Rolit. However, it is still outperformed by BRS with a win ratio between 65% and 70%.

6.5.4 BRS against Paranoid

Table 6.5 shows the performance of BRS against paranoid in Chinese Checkers, Focus, and Rolit. Draws are addressed in a similar manner as in Table 6.4. For the 5,000 ms experiment of Rolit we played the double number of games to reach statistical significance (2,004 games for three players and 2,016 for four players).

For three-player Chinese Checkers, BRS wins above 70% of the games against paranoid. For four and six players, a win ratio of approximately 60% is achieved. In this game, we do not observe a clear performance trend when increasing the thinking time. As expected, paranoid performs better against BRS than max^n did in Chinese Checkers.

Table 6.4: Winning statistics for BRS vs. Max^n.

Chinese Checkers				
Players	Time (ms)	BRS	Max^n	Win Ratio
3	250	818	184	81.6% ± 2.4%
3	1,000	882	120	88.0% ± 2.0%
3	5,000	871	131	86.9% ± 2.1%
4	250	730	278	72.4% ± 2.8%
4	1,000	857	151	85.0% ± 2.2%
4	5,000	846	162	83.9% ± 2.3%
6	250	735	319	72.9% ± 2.7%
6	1,000	793	261	78.7% ± 2.5%
6	5,000	832	222	82.5% ± 2.3%

Focus				
Players	Time (ms)	BRS	Max^n	Win Ratio
3	250	940.7	61.3	93.9% ± 1.5%
3	1,000	952.0	50.0	95.0% ± 1.3%
3	5,000	868.0	134.0	86.6% ± 2.1%
4	250	841.0	167.0	83.4% ± 2.3%
4	1,000	823.3	184.8	81.7% ± 2.4%
4	5,000	818.5	189.5	81.2% ± 2.4%

Rolit				
Players	Time (ms)	BRS	Max^n	Win Ratio
3	250	637.5	364.5	63.6% ± 3.0%
3	1,000	661.5	340.5	66.0% ± 2.9%
3	5,000	650.5	351.5	64.9% ± 3.0%
4	250	690.5	317.5	68.5% ± 2.9%
4	1,000	696.9	311.1	69.1% ± 2.9%
4	5,000	664.5	343.5	65.9% ± 2.9%

In the experiments of BRS against paranoid in Focus, we see that a relatively stable win ratio between 58.0% and 68.2% is achieved. We again may conclude that BRS plays stronger than paranoid.

In Rolit, we see that BRS is weaker in a short time setting for three and four players when playing against paranoid. A possible reason for this is that due to the short thinking time, the paranoid player is not too paranoid yet. With a deeper search, paranoid may become too careful. With 1,000 ms per move, BRS wins about 57% of the games. With 5,000 ms thinking time, the performance of BRS is dropping again to 49.5% ± 2.2% for three players and 52.4% ± 2.2% for four players. A possible explanation for this is that a move in Rolit changes the board significantly and the illegal states have a large influence on this time setting.

Table 6.5: Winning statistics for BRS vs. Paranoid.

Chinese Checkers				
Players	Time (ms)	BRS	Paranoid	Win Ratio
3	250	713	289	71.2% ± 2.8%
3	1,000	763	239	76.1% ± 2.6%
3	5,000	722	280	72.1% ± 2.8%
4	250	594	414	58.9% ± 3.0%
4	1,000	593	415	58.8% ± 3.0%
4	5,000	572	436	56.7% ± 3.1%
6	250	610	444	60.5% ± 3.0%
6	1,000	611	443	60.6% ± 3.0%
6	5,000	596	458	59.1% ± 3.0%

Focus				
Players	Time (ms)	BRS	Paranoid	Win Ratio
3	250	581.3	420.7	58.0% ± 3.1%
3	1,000	683.7	318.3	68.2% ± 2.9%
3	5,000	673.0	329.0	67.2% ± 2.9%
4	250	654.3	353.8	64.9% ± 2.9%
4	1,000	659.3	348.8	65.4% ± 2.9%
4	5,000	609.5	398.5	60.5% ± 3.0%

Rolit				
Players	Time (ms)	BRS	Paranoid	Win Ratio
3	250	293.5	708.5	29.3% ± 2.8%
3	1,000	580.0	422.0	57.9% ± 3.1%
3	5,000	992.5	1011.5	49.5% ± 2.2%
4	250	490.0	518.0	48.6% ± 3.1%
4	1,000	580.2	427.8	57.6% ± 3.1%
4	5,000	1055.5	960.5	52.4% ± 2.2%

6.5.5 BRS vs. Paranoid vs. Max^n

When competing against one type of opponent, one can win the game if the opponent's weakness is discovered. When playing against different kinds of opponents, the game dynamics change. Every opponent has different weak spots which have to be exploited at the same time. Therefore, we matched BRS, paranoid, and max^n against each other in the three-player variant of Chinese Checkers, Focus and Rolit. The results are shown in Table 6.6. Draws are addressed in a similar manner as before. If all algorithms would be equally strong, a win ratio of 33.3% would be expected.

The results show that max^n clearly is the weakest algorithm of the three, having its best performance in Rolit. For Chinese Checkers and Focus, BRS is clearly the best algorithm. In Rolit, the performance of BRS and paranoid are comparable with 250 and 1,000 ms per move. For 5,000 ms, paranoid is the best algorithm.

Table 6.6: Tournament results.

Chinese Checkers				
Time (ms)	BRS	Paranoid	Max^n	BRS Win Ratio
250	582	333	87	58.1% ± 3.1%
1,000	600	324	78	59.9% ± 3.0%
5,000	661	193	163	66.0% ± 2.9%
Focus				
Time (ms)	BRS	Paranoid	Max^n	BRS Win Ratio
250	507.7	431.7	62.7	50.7% ± 3.1%
1,000	619.7	319.7	62.7	61.8% ± 3.0%
5,000	616.7	265.7	119.7	61.5% ± 3.0%
Rolit				
Time (ms)	BRS	Paranoid	Max^n	BRS Win Ratio
250	416.0	393.5	192.5	41.5% ± 3.1%
1,000	399.0	417.5	185.5	39.8% ± 3.0%
5,000	372.0	453.0	177.0	37.1% ± 3.0%

6.6 Chapter Conclusions and Future Research

In this chapter we proposed a new search algorithm called Best-Reply Search (BRS) for deterministic non-cooperative multi-player games with perfect information. The algorithm allows only one opponent to play a counter move. This opponent is the one with the strongest move against the root player. The other players have to pass their turn. Using this approach, more turns of the root player can be searched, resulting in long-term planning. At the same time, some sort of cautiousness is preserved by searching the strongest opponent move.

The first conclusion we may draw is that BRS is able to significantly outperform max^n in Chinese Checkers, Focus, and Rolit. In Chinese Checkers, BRS wins between 72% and 88% of the games. For Focus, BRS wins more than 80% of the games. An impressive 95% win ratio for three players with 1,000 ms per move was achieved. In Rolit, BRS wins approximately 65% of all games.

Our second conclusion is that against paranoid, BRS is significantly stronger in Chinese Checkers and Focus. In these games BRS won approximately 60% across all experiments. In Rolit, BRS did not perform well with 250 ms of thinking time. However, BRS was stronger than paranoid with 1,000 ms, and on equal footing with 5,000 ms of thinking time. A possible reason why BRS is not outperforming paranoid in Rolit is that the board changes significantly with every move.

Our third conclusion is that when playing different kind of opponents, BRS is the strongest algorithm in Chinese Checkers and Focus. In Rolit, BRS was somewhat behind paranoid.

The fourth conclusion we may draw is that increasing the search time generally does not have a negative effect on the performance of BRS (in Chinese Checkers and Focus). This implies that searching illegal positions, which are generated by forcing opponents to pass, does not have a large influence. The possible negative effect is

outbalanced by the larger lookahead. In spite of this negative effect being visible in Rolit, BRS was still competitive with paranoid.

The first direction of future research is the application of Monte-Carlo algorithms. Over the past years, Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006; Coulom, 2007a) has become increasingly popular for letting computers play games. It has been applied successfully in quite some two-player games (e.g., Go (Coulom, 2007a; Gelly and Silver, 2007; Chaslot *et al.*, 2008d), Amazons (Lorentz, 2008; Kloetzer *et al.*, 2009), Hex (Cazenave and Saffidine, 2009), Kriegspiel (Ciancarini and Favini, 2009) and Lines of Action (Winands and Björnsson, 2010)). Moreover, Cazenave (2008) applied MCTS successfully for multi-player Go. Sturtevant (2008a) showed that MCTS outperforms $\max^n$ and paranoid in Chinese Checkers, when given enough time. Nijssen and Winands (2011) proposed a new MCTS enhancement, *progressive history*, that combines progressive bias and the history heuristic. Results showed that the progressive history improved the performance of MCTS in Chinese Checkers and Focus. An interesting experiment would be to compare the playing strength of BRS against an MCTS program in Chinese Checkers and Focus. Furthermore, the BRS principle can be used in MCTS programs as well. This might lead to an improvement in playing strength.

The second direction of future research is variable-depth search (Marsland and Björnsson, 2001). Forward-pruning techniques prune unpromising branches in advance with only a small risk. The most prominent techniques are null moves (Beal, 1989; Goetsch and Campell, 1990), ProbCut (Buro, 1995) and Multi-Cut (Björnsson and Marsland, 2001). A prerequisite of successfully applying these techniques is to have a strong evaluation function. This function should be a good predictor and not suffer from the horizon or odd-even effects. Using these techniques, an even larger lookahead would be possible. Because of the direct succession of MAX and MIN nodes in BRS, we expect that forward-pruning techniques are more effective in BRS than in paranoid. This could give BRS an edge over paranoid.

The third to seventh directions for future research are as follows. (3) Searching illegal positions is not necessary for BRS. Instead, the opponents who are not selected for the counter move could be allowed to play the first move from the static move ordering. This may make BRS applicable to the game of Hearts. (4) BRS should be tested for more domains, such as four-player chess (Lorenz and Tscheuschner, 2006). (5) Because lookahead is important, it would be interesting to test how an algorithm performs which only searches moves by the root player (making it a one-player game, as in the evaluation function for Chinese Checkers by Sturtevant, 2003a). (6) The MP-Mixed algorithm chooses a search method based on the current situation of the game (Zuckerman *et al.*, 2009). BRS may be able to improve the strength of this technique as well. (7) It should be investigated what the applicability of BRS is in non-deterministic games or games with imperfect information.

Chapter 7

Conclusions and Future Research

This thesis investigated how selective-search methods can improve the performance of a game program for a given domain. This led to the formulation of our problem statement in Section 1.3.

> **Problem statement**: *How can we improve selective-search methods in such a way that programs increase their performance in domains of different complexity?*

Rather than testing selective-search methods on one class of game, we chose different classes of games, which all have to be addressed differently. Each class of games represents a level of complexity. Between every level there exists a complexity jump. With a complexity jump the complexity of the game increases significantly, because the mechanism of the game is changed (e.g., a player, chance or imperfect information is added). The domains consisted of deterministic one-, two- and multi-player games with perfect information, and two-player non-deterministic or imperfect-information games. We have posed four research questions that should be answered before we could address the problem statement.

In this chapter, we present the conclusions of the thesis. In Section 7.1 we answer the four research questions one by one. We formulate an answer to the problem statement in Section 7.2. Finally, in Section 7.3 we provide promising directions of future research.

7.1 Conclusions on the Research Questions

The four research questions stated in Chapter 1 concern different classes of games, each with a different level of complexity, i.e., (1) one-player games, (2) two-player games, (3) two-player games with non-deterministic and imperfect-information, and (4) multi-player games. They are dealt with in the following subsections, respectively.

7.1.1 One-Player Games

The traditional approach to deterministic one-player games with perfect information is applying A* or IDA*. These methods have been quite successful in coping with this type of games. The disadvantage of these methods is that they require an admissible heuristic evaluation function. The construction of such a function can be difficult. Since Monte-Carlo Tree Search (MCTS) does not require an admissible heuristic, it may be an interesting alternative. This has led us to the first research question.

> **Research question 1**: *How can we adapt Monte-Carlo Tree Search for a one-player game?*

To answer the first research question, we proposed a new MCTS variant called Single-Player Monte-Carlo Tree Search (SP-MCTS). We adapted MCTS by two modifications resulting in SP-MCTS. The modifications concern (1) the selection strategy and (2) the backpropagation strategy. For testing SP-MCTS, we have chosen the puzzle SameGame as test domain. So far, there does not exist a good admissible heuristic evaluation function for this game.

On the standardized test set of 20 SameGame positions, the manually tuned SP-MCTS method, which invests all search time at the initial position, scored 73,998 points. This was the highest score on the test set at that point of time (2008). The main contribution is therefore that we successfully adapted MCTS for a one-player game. Inspired by our approach, two other Monte-Carlo-based approaches, Nested Monte-Carlo Search (Cazenave, 2009) and Heuristically Guided Swarm Tree Search (Edelkamp *et al.*, 2010), broke our record subsequently. At the time of publishing this thesis SP-MCTS, with parameters tuned by the Cross-Entropy Method and with time equally distributed over the consecutive positions, scored 78,012 points on the test set, which is currently the third highest score (2010). Thus, answering research question 1, we have shown that MCTS is applicable to a one-player deterministic perfect-information game. Our variant, SP-MCTS, is able to achieve good results in the game of SameGame. SP-MCTS is a worthy alternative for puzzles where a good admissible estimator cannot be found.

7.1.2 Two-Player Games

Ideally, a search method is able to prove that a move is the optimal one for a given game. The game is solved if this is achieved. In the last years quite some deterministic two-player games with perfect information have been solved. A search method specially designed as mate-solver is Proof-Number (PN) search. PN search is efficient in searching game trees with a non-uniform branching factor. However, PN search has to expand nodes until the end of the game is reached. Moreover, for quite some games, endgame databases played a substantial role in solving. When using endgame databases, branches entering the database can be pruned. This has led us to the second research question.

> **Research question 2**: *How can we solve a two-player game by using Proof-Number search in combination with endgame databases?*

We examine the tradeoff between time spent on PN search and time spent on creating endgame databases when solving the game of Fanorona. This game has a material-based theme with a state-space complexity similar to checkers. Endgame-database statistics show that (1) the player to move has an advantage and (2) that a draw can often be achieved in spite of having fewer pieces than the opponent. The optimal endgame-database size for the 3×9, 5×5 and 7×5 Fanorona variants are 3, 4 and 5 pieces, respectively. We conclude that the optimal database size is located at the point where the time required for database construction and the time required for solving by PN search are of the same order. Our main result is that the game of Fanorona (5×9) has been weakly solved and is drawn when both players play optimally, adding Fanorona to the list of solved games (cf. Van den Herik *et al.*, 2002). This result was achieved by a well-chosen combination of the PN-search variant PN^2 and all endgame databases up to 7 pieces.

Finally, we remark that we were the first to compute and analyze these endgame databases for Fanorona. From these experiments we offer two additional conclusions. The first conclusion we draw is that the optimal endgame-database size for Fanorona (5×9) is 6 or 7 pieces. The time required for solving Fanorona with 7 pieces was less than the time required for creating the 7-piece endgame databases. The second conclusion we draw is that White is able to force a win on board sizes with one side equal to 3. We conjecture that for boards where both sides have at least size 5, White does not have this advantage for the majority of cases.

7.1.3 Two-Player Games with Non-Determinism and Imperfect Information

In variable-depth search, branches can be pruned if they seem unpromising (forward pruning), or extended if the branches are promising (search extensions). There exist several successful forward-pruning techniques for the $\alpha\beta$ algorithm. For two-player games with non-determinism or imperfect information expectimax may be used. Expectimax adds chance nodes to the search tree. There are, however, no forward-pruning techniques available for chance nodes. This has led us to the third research question.

> **Research question 3**: *How can we perform forward pruning at chance nodes in the expectimax framework?*

For answering the third research question, we have proposed the forward-pruning technique ChanceProbCut for expectimax. This technique is the first in its kind to forward prune at chance nodes. ChanceProbCut is inspired by the $\alpha\beta$ forward-pruning technique ProbCut (Buro, 1995). ChanceProbCut estimates values of chance events based on shallow searches. Based on the correlation between evaluations obtained from searches at different depths, ChanceProbCut prunes chance events in advance if the result of the chance node probably falls outside the search window. Two non-deterministic games (Dice and ChanceBreakthrough) and a game of imperfect information (Stratego) served as test domains. Experiments revealed that ChanceProbCut is able to reduce the size of the game tree significantly without a loss of decision quality in Stratego, Dice, and ChanceBreakthrough. A safe node

reduction of between 30% and 85% is achieved across all games. Thus, ChanceProbCut finds a good move faster in the expectimax framework, while not affecting the playing strength. The gained time may be invested in a deeper search. Selfplay experiments in Stratego and Dice showed that there is a small but relevant improvement in playing strength. In ChanceBreakthrough, though, a significant increase in performance was measured. ChanceProbCut was able to win 54.4% on 4,000 games.

Thus, answering research question 3, the proposed forward-pruning technique ChanceProbCut improves the search performance in non-deterministic games and games with imperfect information.

7.1.4 Multi-Player Games

In deterministic two-player games with perfect information, the majority of research focused on the $\alpha\beta$ algorithm. For deterministic multi-player games with perfect information, the choice of algorithm is not as straightforward. The two main algorithms are called $\max^n$ and paranoid, both approaching the problem from a different angle. Max^n assumes that every player tries to maximize the own score, while paranoid assumes that all opponents form a coalition against the root player. However, these assumptions have drawbacks. This has led us to the fourth research question.

> **Research question 4**: *How can we improve search for multi-player games?*

For answering the fourth research question, we proposed a new search algorithm, called Best-Reply Search (BRS), for deterministic non-cooperative multi-player games with perfect information. This algorithm allows only one opponent to play a counter move. This opponent is the one with the strongest move against the root player. The other players have to pass their turn. Using this approach, more turns of the root player can be searched, resulting in long-term planning. At the same time, some sort of cautiousness is preserved by searching the strongest opponent move.

We have chosen three deterministic multi-player games of perfect information, i.e., Chinese Checkers, Focus, and Rolit. BRS is able to significantly outperform $\max^n$ in these games, with a win ratio of between 65% and 95%. Against paranoid, BRS is significantly stronger in Chinese Checkers and Focus, with win ratios of between 57% and 71%. In Rolit, BRS and paranoid are on equal footing. When playing different kind of opponents at the same time, BRS is the strongest algorithm in Chinese Checkers and Focus. In Rolit, BRS was somewhat behind paranoid. Increasing the search time generally does not have a negative effect on the performance of BRS. This implies that searching illegal positions, which are generated by forcing opponents to pass, does not have a large influence. The possible negative effect is outbalanced by the larger lookahead.

Thus, answering research question 4, the proposed search algorithm BRS is able to significantly outperform both established algorithms, the $\max^n$ and paranoid algorithm.

7.2 Conclusion on the Problem Statement

After answering all four research questions, we are now able to provide an answer to the problem statement.

> **Problem statement**: *How can we improve selective-search methods in such a way that programs increase their performance in domains of different complexity?*

Taking the answers to the research questions above into account we see that there are several ways to improve selective-search methods. We can summarize these in four points. First, Single-Player Monte-Carlo Tree Search balances exploitation and exploration such that it is a worthy alternative for one-player games where a good admissible estimator cannot be found. Second, PN search with endgame databases, which prefers narrow subtrees above wide ones, was able to prove that the game-theoretic value of the two-player game Fanorona is a draw. Third, ChanceProbCut is able to forward prune at chance nodes in two-player games with non-determinism or imperfect information. Fourth, in non-cooperative deterministic multi-player games with perfect information, Best-Reply Search achieves long-term planning by assuming that only one opponent is allowed to play a counter move.

7.3 Recommendations for Future Research

The research presented in this thesis indicates the following areas of future research.

1. **Improving SP-MCTS.** We mention three possible enhancements in SP-MCTS. (1) Knowledge can be included in the selection mechanism with RAVE (Gelly and Silver, 2007) or progressive widening (Coulom, 2007a; Chaslot *et al.*, 2008d). (2) We demonstrated that combining small searches can achieve better scores than one large search. However, there is no information shared between the searches. This can be achieved by using a transposition table, which is not cleared at the end of a small search. (3) Regular root parallelization should be investigated to take advantage of multi-processor architectures.

2. **Improving PN-Search.** The time for solving may be reduced significantly by using Evaluation-Function Based Proof-Number Search (EF-PN) (Winands and Schadd, 2011). EF-PN is a general framework for employing a traditional evaluation function in PN search. The search is directed to branches where the evaluation function indicates a promising situation. For Fanorona, the material on the board may be used as an evaluation function.

3. **Improving ChanceProbCut.** We propose three directions for improving ChanceProbCut. (1) ChanceProbCut uses a number of linear regression models to predict the value at depth d using a value of depth $d-R$. For improving the effectiveness of ChanceProbCut, additional linear regression models for different depths may be used. (2) The regression parameters and cut-threshold t

can be bootstrapped according to the game phase. (3) A successor of ProbCut exists, called Multi-ProbCut (Buro, 2000). This technique could also be adapted for chance nodes (i.e., Multi-ChanceProbCut).

4. **Testing and improving BRS.** Sturtevant (2008a) showed that MCTS outperforms $\max^n$ and paranoid in Chinese Checkers, when given enough time. For further testing BRS, an interesting experiment would be to compare the playing strength of BRS against an MCTS program in Chinese Checkers. Furthermore, the BRS principle can be used in MCTS programs as well. This might lead to an improvement in playing strength.

   For improving BRS, a possible direction is variable-depth search (Marsland and Björnsson, 2001). The most prominent forward-pruning techniques are null moves (Beal, 1989; Goetsch and Campell, 1990), ProbCut (Buro, 1995) and Multi-Cut (Björnsson and Marsland, 2001). Using these techniques, an even larger lookahead would be possible. Because of the direct succession of MAX and MIN nodes in BRS, we expect that forward-pruning techniques are more effective in BRS than in paranoid. This could give BRS an edge over paranoid.

   We provide three further approaches to improve BRS. (1) Searching illegal positions is not necessary for BRS. Instead, the opponents who are not selected for the counter move could be allowed to play the first move from the static move ordering. This may make BRS applicable to games like Hearts. (2) Because lookahead is important, it would be interesting to test how an algorithm performs that only searches moves by the root player (making it a one-player game, as in the evaluation function for Chinese Checkers by Sturtevant, 2003a). (3) The MP-Mixed algorithm chooses a search method based on the current situation of the game (Zuckerman *et al.*, 2009). BRS may be able to improve the strength of this technique as well.

5. **Application to other domains.** All our proposed enhancements and algorithms can be tested in other domains. These domains include classes of games with corresponding complexity levels, but also game classes which have not been covered in this research. We mention one-player games with non-determinism, two-player games with non-determinism and imperfect information, and multi-player games with non-determinism and/or imperfect information.

References

Adachi, H., Kamekawa, H., and Iwata, S. (1987). Shogi on n × n Board is Complete in Exponential Time. *Transactions of the Institute of Electronics, Information and Communication Engineers*, Vol. J70-D, No. 10, pp. 1843–1852. In Japanese. [2]

Adelson-Velskiy, G. M., Arlazarov, V. L., and Donskoy, M. V. (1975). Some Methods of Controlling the Tree Search in Chess Programs. *Artificial Intelligence*, Vol. 6, No. 4, pp. 361–371. [69]

Akl, S. G. and Newborn, M. M. (1977). The Principal Continuation and the Killer Heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466–473, ACM Press, New York, NY, USA. [15, 77, 101]

Allis, L. V. (1988). *A Knowledge-Based Approach of Connect-Four. The Game is Solved: White Wins.* M.Sc. thesis, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands. [7, 42]

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence.* Ph.D. thesis, Department of Computer Science, Rijksuniversiteit Limburg, Maastricht, The Netherlands. [27, 42, 47, 53]

Allis, L. V. and Schoo, P. N. A. (1992). Qubic Solved Again. *Heuristic Programming in Artificial Intelligence 3: The Third Computer Olympiad* (eds. H. J. van den Herik and L. V. Allis), pp. 192–204, Ellis Horwood Limited, Chichester, United Kingdom. [42]

Allis, L. V., Herik, H. J. van den, and Herschberg, I. S. (1991). Which Games Will Survive? *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad* (eds. D. N. L. Levy and D. F. Beal), pp. 232–243, Ellis Horwood Limited, Chichester, United Kingdom. [60]

Allis, L. V., Herik, H. J. van den, and Huntjes, M. P. H. (1993). Go-Moku and Threat-Space Search. Technical Report CS 93–02, Department of Computer Science, Faculty of General Sciences, Rijksuniversiteit Limburg, Maastricht, The Netherlands. [42]

Allis, L. V., Meulen, M. van der, and Herik, H. J. van den (1994). Proof-Number Search. *Artificial Intelligence*, Vol. 66, No. 1, pp. 91–124. [6, 8, 43, 50, 52]

Allis, L. V., Herik, H. J. van den, and Huntjes, M. P. H. (1996). Go-Moku Solved by New Search Techniques. *Computational Intelligence*, Vol. 12, No. 1, pp. 7–23. [42, 43]

Anshelevich, V. V. (2000). The Game of Hex: An Automatic Theorem Proving Approach to Game Programming. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00)*, pp. 189–194, AAAI Press, Menlo Park, CA, USA. [42]

Arneson, B., Hayward, R. B., and Henderson, P. (2010). Monte Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 251–258. [6]

Arts, A. F. C. (2010). Competitive Play in Stratego. M.Sc. thesis, Maastricht University, Maastricht, The Netherlands. [75]

Ash, R. B. and Bishop, R. L. (1972). Monopoly® as a Markov process. *Mathematics Magazine*, Vol. 45, No. 1, pp. 26–29. [4]

Audibert, J.-Y. and Bubeck, S. (2009). Minimax Policies for Adversarial and Stochastic Bandits. *Proceedings of the 22nd Annual Conference on Learning Theory (COLT 2009)* (eds. S. Dasgupta and A. Klivans), pp. 217–226, Omnipress, Eastbourne, United Kingdom. [20]

Bakkes, S. C. J. (2010). *Rapid Adaptation of Video Game AI*. Ph.D. thesis, Tilburg Centre for Cognition and Communication, Universiteit van Tilburg, Tilburg, The Netherlands. [4, 5]

Ballard, B. W. (1983). The *-Minimax Search Procedure for Trees Containing Chance Nodes. *Artificial Intelligence*, Vol. 21, No. 3, pp. 327–350. [66, 67, 71, 77]

Beal, D. F. (1989). Experiments with the Null Move. *Advances in Computer Chess 5* (ed. D. F. Beal), pp. 65–89, Elsevier Science Publishers, Amsterdam, The Netherlands. [8, 63, 69, 101, 108, 114]

Beal, D. F. and Smith, M. C. (1996). Multiple Probes of Transposition Tables. *ICCA Journal*, Vol. 19, No. 4, pp. 227–233. [18]

Beal, D. F. and Smith, M. C. (2000). Temporal Difference Learning for Heuristic Search and Game Playing. *Information Sciences*, Vol. 122, No. 1, pp. 3–21. [33]

Bell, R. C. (1980). *Board and Table Games from Many Civilizations*. Dover Publications, New York, NY, USA. [1, 43]

Benthem, J. van (2001). Games in Dynamic-Epistemic Logic. *Proceedings of LOFT-4* (eds. G. Bonanno and W. van der Hoek), Vol. 53, No. 4 of *Bulletin of Economic Research*, pp. 219–248. [2]

Benthem, J. van, Girard, P., and Roy, O. (2008). Everything Else Being Equal: A Modal Logic for Ceteris Paribus Preferences. *Journal of Philosophical Logic*, Vol. 38, No. 1, pp. 83–125. [90]

Berlekamp, E. R., Conway, J. H., and Guy, R. K. (1982). *Winning Ways*, Vol. 1–2. Academic Press, Inc., London, United Kingdom. [2]

Biedl, T. C., Demaine, E. D., Demaine, M. L., Fleischer, R., Jacobsen, L., and Munro, J. I. (2002). The Complexity of Clickomania. *More Games of No Chance, Proceedings of the 2002 MSRI Workshop on Combinatorial Games* (ed. R. J. Nowakowski), Vol. 42 of *MSRI Publications*, pp. 389–404, Cambridge University Press, Cambridge, United Kingdom. [27, 28]

Billings, D. (2007). Personal Communication, University of Alberta, Canada. [28]

Billings, D., Papp, D., Schaeffer, J., and Szafron, D. (1998a). Poker as Testbed for AI Research. *Canadian Conference on AI* (eds. R. E. Mercer and E. Neufeld), Vol. 1418 of *Lecture Notes in Computer Science (LNCS)*, pp. 228–238, Springer-Verlag, Berlin, Germany. [4, 64]

Billings, D., Papp, D., Schaeffer, J., and Szafron, D. (1998b). Opponent Modeling in Poker. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)* (eds. J. Mostow, C. Rich, and B. Buchanan), pp. 493–499, AAAI Press, Menlo Park, CA, USA. [4, 64]

Billings, D., Peña, L., Schaeffer, J., and Szafron, D. (1999). Using Probabilistic Knowledge and Simulation to Play Poker. *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pp. 697–703, AAAI Press, Menlo Park, CA, USA. [4, 18]

Bjarnason, R., Tadepalli, P., and Fern, A. (2007). Searching Solitaire in Real Time. *ICGA Journal*, Vol. 30, No. 3, pp. 131–142. [4]

Björnsson, Y. (2002). *Selective Depth-First Game-Tree Search*. Ph.D. thesis, Department of Computing Science, University of Alberta, Edmonton, Canada. [6, 69]

Björnsson, Y. and Finnsson, H. (2009). CadiaPlayer: A Simulation-Based General Game Player. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 1, No. 1, pp. 4–15. [76]

Björnsson, Y. and Marsland, T. A. (1999). Multi-Cut Alpha-Beta Pruning. *Computers and Games (CG 1998)* (eds. H. J. van den Herik and H. Iida), Vol. 1558 of *Lecture Notes in Computer Science (LNCS)*, pp. 15–24, Springer-Verlag, Berlin, Germany. [63, 69]

Björnsson, Y. and Marsland, T. A. (2000). Risk Management in Game-Tree Pruning. *Information Sciences*, Vol. 122, No. 1, pp. 23–41. [63]

Björnsson, Y. and Marsland, T. A. (2001). Multi-Cut $\alpha\beta$-Pruning in Game-Tree Search. *Theoretical Computer Science*, Vol. 252, Nos. 1–2, pp. 177–196. [8, 63, 69, 101, 108, 114]

Boer, V. de (2007). Invincible. A Stratego Bot. M.Sc. thesis, Delft University of Technology, Delft, The Netherlands. [63, 77]

Boer, V. de, Rothkrantz, L. J. M., and Wiggers, P. (2008). Invincible: A Stratego Bot. *International Journal of Intelligent Games & Simulation*, Vol. 5, No. 1, pp. 22–28. [75]

Bono, E. de (1968). *The Five-Day Course in Thinking*. Penguin, New York, NY, USA. [1]

Borel, É. (1921). La Théorie du Jeu et les Equations Intégrales à Noyau Symétrique Gauche. *Comptes Rendus de l'Académie des Sciences*, Vol. 173, pp. 1304–1308. In French. Translated by L. J. Savage (1953) as The Theory of Play and Integral Equations with Skew Symmetric Kernels. *Econometria*, Vol. 21, No.1, pp. 97–100. [2]

Borsboom, J., Saito, J.-T., Chaslot, G. M. J.-B., and Uiterwijk, J. W. M. H. (2007). A Comparison of Monte-Carlo Methods for Phantom Go. *Proceedings of the 19th BeNeLux Conference on Artificial Intelligence (BNAIC'07)* (eds. M. M. Dastani and E. de Jong), pp. 57–64, University of Utrecht, Utrecht, The Netherlands. [3]

Bouzy, B. (2005). Associating Domain-Dependent Knowledge and Monte-Carlo Approaches within a Go Program. *Information Sciences, Heuristic Search and Computer Game Playing IV*, Vol. 175, No. 4, pp. 247–257. [20, 31]

Bouzy, B. and Helmstetter, B. (2003). Monte-Carlo Go Developments. *Advances in Computer Games (ACG 10)* (eds. H. J. van den Herik, H. Iida, and E. A. Heinz), pp. 159–174, Kluwer Academic, Dordrecht, The Netherlands. [18]

Breuker, D. M. (1998). *Memory versus Search in Games*. Ph.D. thesis, Department of Computer Science, Maastricht University, Maastricht, The Netherlands. [18, 101]

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (1994). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 4, pp. 183–193. [18]

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (1996). Replacement Schemes and Two-Level Tables. *ICCA Journal*, Vol. 19, No. 3, pp. 175–180. [18]

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2000). Solving 8×8 Domineering. *Theoretical Computer Science*, Vol. 230, Nos. 1–2, pp. 195–206. [42]

Breuker, D. M., Herik, H. J. van den, Uiterwijk, J. W. H. M., and Allis, L. V. (2001a). A Solution to the GHI Problem for Best-First Search. *Theoretical Computer Science*, Vol. 252, Nos. 1–2, pp. 121–149. [60]

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2001b). The PN^2-Search Algorithm. *Advances in Computer Games 9* (eds. H. J. van den Herik and B. Monien), pp. 115–132, Maastricht University, Maastricht, The Netherlands. [53]

Brügmann, B. (1993). Monte Carlo Go. Technical report, Physics Department, Syracuse University, Syracuse, NY, USA. [18, 22]

Bullock, N. (2002). Domineering: Solving Large Combinatorial Search Spaces. *ICGA Journal*, Vol. 25, No. 2, pp. 67–84. [42]

Buro, M. (1995). ProbCut: An Effective Selective Extension of the Alpha-Beta Algorithm. *ICCA Journal*, Vol. 18, No. 2, pp. 71–76. [6, 8, 63, 64, 69, 70, 71, 101, 108, 111, 114]

Buro, M. (2000). Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello. *Games in AI Research* (eds. H. J. van den Herik and H. Iida), pp. 77–96, Maastricht University, Maastricht, The Netherlands. [8, 63, 69, 88, 114]

Buro, M. (2003). The Evolution of Strong Othello Programs. *Entertainment Computing - Technology and Applications* (eds. R. Nakatsu and J. Hoshino), Vol. 112 of *IFIP Advances in Information and Communication Technology*, pp. 81–88, Kluwer Academic Publishers, Dordrecht, The Netherlands. [100, 101]

Campbell, M. (1985). The Graph-History Interaction: On Ignoring Position History. *Proceedings of the 1985 ACM Annual Conference on the Range of Computing: Mid-80's Perspective*, pp. 278–280, ACM Press, New York, NY, USA. [17, 60]

Carmel, D. and Markovitch, S. (1993). Learning Models of Opponent's Strategies in Game Playing. *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning* (eds. R. Levinson and S. Epstein), pp. 140–147, AAAI Press, Menlo Park, CA, USA. [4, 64]

Carter, R. G. (2007). *An Investigation into Tournament Poker Strategy using Evolutionary Algorithms*. Ph.D. thesis, School of Informatics, University of Edinburgh, Edinburgh, United Kingdom. [4, 63]

Cazenave, T. (2006). A Phantom-Go Program. *Advances in Computer Games (ACG 11)* (eds. H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers), Vol. 4250 of *Lecture Notes in Computer Science (LNCS)*, pp. 120–125, Springer-Verlag, Berlin Heidelberg, Germany. [3]

Cazenave, T. (2008). Multi-player Go. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 50–59, Springer-Verlag, Berlin, Germany. [108]

Cazenave, T. (2009). Nested Monte-Carlo Search. *Proceedings of the Twenty-first International Joint Conferences on Artificial Intelligence (IJCAI-09)* (ed. C. Boutilier), pp. 456–461, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [28, 34, 40, 110]

Cazenave, T. and Jouandeau, N. (2007). On the Parallelization of UCT. *Proceedings of the Computer Games Workshop 2007 (CGW 2007)* (eds. H. J. van den Herik,

J. W. H. M. Uiterwijk, M. H. M. Winands, and M. P. D. Schadd), pp. 93–101, Maastricht University, Maastricht, The Netherlands. [22, 23]

Cazenave, T. and Saffidine, A. (2009). Utilisation de la Recherche Arborescente Monte-Carlo au Hex. *Revue d'Intelligence Artificielle*, Vol. 23, Nos. 2–3, pp. 183–202. In French. [6, 108]

Chaslot, G. M. J.-B. (2010). *Monte-Carlo Tree Search*. Ph.D. thesis, Department of Knowledge Engineering, Maastricht University, Maastricht, The Netherlands. [19, 20, 22, 31]

Chaslot, G. M. J.-B., Jong, S. de, Saito, J.-T., and Uiterwijk, J. W. H. M. (2006a). Monte-Carlo Tree Search in Production Management Problems. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence (BNAIC'06)* (eds. P.-Y. Schobbens, W. Vanhoof, and G. Schwanen), pp. 91–98, University of Namur, Namur, Belgium. [19, 26, 30]

Chaslot, G. M. J.-B., Saito, J.-T., Bouzy, B., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2006b). Monte-Carlo Strategies for Computer Go. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence (BNAIC'06)* (eds. P.-Y. Schobbens, W. Vanhoof, and G. Schwanen), pp. 83–90, University of Namur, Namur, Belgium. [6, 19, 20, 21, 30, 32]

Chaslot, G. M. J.-B., Hoock, J.-B., Rimmel, A., Teytaud, O., Lee, C.-S., Wang, M.-H., Tsai, S.-R., and Hsu, S.-C. (2008a). Human-Computer Go Revolution 2008. *ICGA Journal*, Vol. 31, No. 3, pp. 179–185. [6]

Chaslot, G. M. J.-B., Winands, M. H. M., and Herik, H. J. van den (2008b). Parallel Monte-Carlo Tree Search. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 60–71, Springer-Verlag, Berlin, Germany. [22, 23, 32]

Chaslot, G. M. J.-B., Winands, M. H. M., Szita, I., and Herik, H. J. van den (2008c). Cross-Entropy for Monte-Carlo Tree Search. *ICGA Journal*, Vol. 31, No. 3, pp. 145–156. [33, 34]

Chaslot, G. M. J.-B., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bouzy, B. (2008d). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357. [6, 19, 21, 22, 40, 108, 113]

Chauvicourt, J. and Chauvicourt, S. (1980). *Le Fanorona – Jeu National Malgache*. Nouvelle Imprimerie des Arts Graphiques, Tananarive, Madagascar. In French. [43]

Chen, K.-H. and Zhang, P. (2008). Monte-Carlo Go with Knowledge-Guided Simulations. *ICGA Journal*, Vol. 31, No. 2, pp. 67–76. [20, 31]

Ciancarini, P. and Favini, G. P. (2007). A Program to Play Kriegspiel. *ICGA Journal*, Vol. 30, No. 1, pp. 3–24. [3, 63]

Ciancarini, P. and Favini, G. P. (2009). Monte Carlo Tree Search Techniques in the Game of Kriegspiel. *Proceedings of the Twenty-first International Joint Conferences on Artificial Intelligence (IJCAI-09)* (ed. C. Boutilier), pp. 474–479, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [108]

Condon, A. (1992). The Complexity of Stochastic Games. *Information and Computation*, Vol. 96, No. 2, pp. 203–224. [64]

Coulom, R. (2007a). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Computers and Games (CG 2006)* (eds. H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers), Vol. 4630 of *Lecture Notes in Computer Science (LNCS)*, pp. 72–83, Springer-Verlag, Heidelberg, Germany. [6, 19, 20, 21, 30, 31, 32, 40, 108, 113]

Coulom, R. (2007b). Computing "Elo Ratings" of Move Patterns in the Game of Go. *ICGA Journal*, Vol. 30, No. 4, pp. 199–208. [22]

Crawford, C. (1984). *The Art of Computer Game Design*. McGraw-Hill/Osborne Media, New York, NY, USA. [5]

Culberson, J. C. and Schaeffer, J. (1998). Pattern Databases. *Computational Intelligence*, Vol. 14, No. 3, pp. 318–334. [29]

Dekker, S. T., Herik, H. J. van den, and Herschberg, I. S. (1990). Perfect Knowledge Revisited. *Artificial Intelligence*, Vol. 43, No. 1, pp. 111–123. [50]

Demaine, E. D. and Hearn, R. A. (2001). Playing Games with Algorithms: Algorithmic Combinatorial Game Theory. *Proceedings of the 26th Symposium on Mathematical Foundations in Computer Science (MFCS 2001 Mariánské Lazné)* (eds. J. Sgall, A. Pultr, and P. Kolman), Vol. 2136 of *Lecture Notes in Computer Science (LNCS)*, pp. 18–32, Springer-Verlag, Berlin, Germany. Reprinted (2009) in Games of No Chance 3, Proceedings of the 2005 MSRI Workshop on Combinatorial Games (eds. M. H. Albert and K. J. Nowakowski), Vol. 56 of MSRI Publications, pp. 3–56., Cambridge University Press, Cambridge, United Kingdom. [2]

Demaine, E. D., Hohenberger, S., and Liben-Nowell, D. (2003). Tetris is Hard, Even to Approximate. *Proceedings of the 9th International Computing and Combinatorics Conference (COCOON 2003)* (eds. T. Warnow and B. Zhu), Vol. 2697 of *Lecture Notes in Computer Science (LNCS)*, pp. 351–363, Springer-Verlag, Berlin, Germany. [4]

District Court of Oregon (2005). *Estate of Gunter Sigmund Elkan, vs. Hasbro, INC. et al.* Case No. 04-1344-KI, Portland, OR, USA. [74]

Ditmarsch, H. van (2000). *Knowledge Games*. Ph.D. thesis, Faculty of Mathematics and Natural Sciences, University of Groningen, Groningen, The Netherlands. [63]

Ditmarsch, H. van (2001). Knowledge Games. *Bulletin of Economic Research*, Vol. 53, No. 4, pp. 249–273. [63]

Doberkat, E.-E., Hasselbring, W., and Pahl, C. (1996). Investigating Strategies for Cooperative Planning of Independent Agents through Prototype Evaluation. *Proceedings of the First International Conference on Coordination Languages and Models (COORDINATION '96)* (eds. P. Ciancarini and C. Hankin), Vol. 1061 of *Lecture Notes in Computer Science (LNCS)*, pp. 416–419, Springer-Verlag, Berlin, Germany. [4]

Donkers, H. H. L. M. (2003). *Nosce Hostem: Searching with Opponent Models*. Ph.D. thesis, Department of Computer Science, Maastricht University, Maastricht, The Netherlands. [4, 64]

Edelkamp, S., Kissmann, P., Sulewski, D., and Messerschmidt, H. (2010). Finding the Needle in the Haystack with Heuristically Guided Swarm Tree Search. *Multikonferenz Wirtschaftsinformatik 2010* (eds. M. Schumann, L. M. Kolbe, M. H. Breitner, and A. Frerichs), pp. 2295–2308, Universitätsverlag Göttingen, Göttingen, Germany. [29, 34, 40, 110]

Emde Boas, P. van (2002). Computational Models of Games. *Lecture Notes*, The Institute for Logic, Language and Computation (ILLC), University of Amsterdam, Amsterdam, The Netherlands. [2]

Emde Boas, P. van (2003). Games, Complexity and Interaction: The Role of Games in Computer Science. *Practical Foundations of Business System Specifications* (eds. H. Kilov and K. Baclavski), pp. 313–328, Springer-Verlag, Berlin Heidelberg, Germany. [7]

Enzenberger, M. and Müller, M. (2010). A Lock-free Multithreaded Monte-Carlo Tree Search Algorithm. *Advances in Computer Games (ACG 2009)* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science (LNCS)*, pp. 14–20, Springer-Verlag, Berlin, Germany. [23]

Felner, A., Zahavi, U., Schaeffer, J., and Holte, R. C. (2005). Dual Lookups in Pattern Databases. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)* (eds. L. P. Kaelbling and A. Saffiotti), pp. 103–108, Professional Book Center, Denver, CO, USA. [29]

Fraenkel, A. S. and Lichtenstein, D. (1981). Computing a Perfect Strategy for n×n Chess Requires Time Exponential in n. *Journal of Combinatorial Theory, Series A*, Vol. 31, No. 2, pp. 199–214. [2]

Frank, A. (1990). Brute Force Search in Games of Imperfect Information. *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad* (eds. D. N. L. Levy and D. F. Beal), pp. 204–209, Ellis Herwood, Chichester, United Kingdom. [64]

Frank, A. and Basin, D. (1998). Search in Games with Incomplete Information: A Case Study using Bridge Card Play. *Artificial Intelligence*, Vol. 100, Nos. 1–2, pp. 87–123. [64]

Frayn, C. M. (2005). An Evolutionary Approach to Strategies for the Game of Monopoly®. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG 2005)* (eds. G. Kendall and S. M. Lucas), pp. 66–72, IEEE press, Piscataway, NJ, USA. [4]

Fukushima, K. (1975). Cognitron: A Self-Organizing Multilayered Neural Network. *Biological Cybernetics*, Vol. 20, Nos. 3–4, pp. 121–136. [3]

Gasser, R. U. (1995). *Harnessing Computational Resources for Efficient Exhaustive Search*. Ph.D. thesis, Department of Computer Science, Swiss Federal Institute of Technology, Zürich, Switserland. [42]

Gasser, R. U. (1996). Solving Nine Men's Morris. *Computational Intelligence*, Vol. 12, No. 1, pp. 24–41. [7]

Gelly, S. and Silver, D. (2007). Combining Online and Offline Knowledge in UCT. *Proceedings of the 24th International Conference on Machine Learning (ICML '07)* (ed. Z. Ghahramani), pp. 273–280, ACM Press, New York, NY, USA. [6, 20, 22, 40, 108, 113]

Gelly, S. and Wang, Y. (2006). Exploration Exploitation in Go: UCT for Monte-Carlo Go. *Neural Information Processing Systems Conference On-line Trading of Exploration and Exploitation Workshop*. [20]

Gelly, S., Wang, Y., Munos, R., and Teytaud, O. (2006). Modification of UCT with Patterns in Monte-Carlo Go. Technical Report 6062, INRIA, Orsay Cedex, France. [20, 31]

Ginsberg, M. L. (1996). Partition Search. *Proceedings of the 13^{th} National Conference on Artificial Intelligence (AAAI-96)*, Vol. 1, pp. 228–233, AAAI Press, Menlo Park, CA, USA. [64]

Ginsberg, M. L. (1999). GIB: Steps Toward an Expert-Level Bridge-Playing Program. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)* (ed. T. Dean), pp. 584–589, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [4]

Goetsch, G. and Campell, M. S. (1990). Experiments with the Null-move Heuristic. *Computers, Chess, and Cognition* (eds. T. A. Marsland and J. Schaeffer), pp. 159–168, Springer-Verlag, New York, NY, USA. [63, 69, 101, 108, 114]

Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt Chess Program. *Proceedings of the AFIPS Fall Joint Computer Conference 31*, pp. 801–810. Reprinted (1988) in *Computer Chess Compendium* (ed. D. N. L. Levy), pp. 56–66. B. T. Batsford Ltd., London, United Kingdom. [17, 77, 101]

Groot, A. D. de (1965). *Thought and Choice in Chess*. Mouton Publishers, The Hague - Paris - New York. [63, 69]

Guy, R. K. (1996). What is a Game? *Games of No Chance, Proceedings of the 1994 MSRI Workshop on Combinatorial Games* (ed. K. J. Nowakowski), Vol. 29 of *MSRI Publications*, pp. 43–60, Cambridge University Press, Cambridge, United Kingdom. [2]

Handscomb, K. (2001). 8×8 Game Design Competition: The Winning Game: Breakthrough ...and two other Favorites. *Abstract Games Magazine*, No. 7, pp. 8–9. [76]

Harel, D. (1984). Dynamic Logic. *Handbook of Philosophical Logic* (eds. D. M. Gabbay and F. Guenther), Vol. 2: Extensions of Classical Logic, pp. 497–604, Springer-Verlag, Berlin, Germany. [90]

Hartmann, D. (1988). Butterfly Boards. *ICCA Journal*, Vol. 11, Nos. 2–3, pp. 64–71. [16]

Hart, P. E., Nielson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernatics*, Vol. SSC–4, No. 2, pp. 100–107. [4, 19, 25, 29]

Hauk, T. (2004). Search in Trees with Chance Nodes. M.Sc. thesis, University of Alberta, Edmonton, Canada. [66, 67, 76, 77, 83, 85]

Hauk, T., Buro, M., and Schaeffer, J. (2006a). Rediscovering *-Minimax Search. *Computers and Games (CG 2004)* (eds. H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu), Vol. 3846 of *Lecture Notes in Computer Science (LNCS)*, pp. 35–50, Springer-Verlag, Berlin, Germany. [66, 71, 85]

Hauk, T., Buro, M., and Schaeffer, J. (2006b). *-Minimax Performance in Backgammon. *Computers and Games (CG 2004)* (eds. H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu), Vol. 3846 of *Lecture Notes in Computer Science (LNCS)*, pp. 51–66, Springer-Verlag, Berlin, Germany. [67, 88]

Hearn, R. A. (2009). Amazons, Konane, and Cross Purposes are PSPACE-complete. *Games of No Chance 3, Proceedings of the 2005 MSRI Workshop on Combinatorial Games* (eds. M. H. Albert and K. J. Nowakowski), Vol. 56 of *MSRI Publications*, pp. 287–306, Cambridge University Press, Cambridge, United Kingdom. [2]

Heinz, E. A. (1999). Endgame Databases and Efficient Index Schemes. *ICCA Journal*, Vol. 22, No. 1, pp. 22–32. [8, 50]

Helmstetter, B. (2007). *Analyses de Dépendances et Méthodes de Monte-Carlo dans les Jeux de Réflexion*. Ph.D. thesis, Laboratoire d'Informatique Avancée de Saint-Denis, Université Paris 8, Paris, France. In French. [19]

Herik, H. J. van den and Herschberg, I. S. (1985). The Construction of an Omniscient Endgame Database. *ICCA Journal*, Vol. 8, No. 2, pp. 66–87. [8, 48, 50]

Herik, H. J. van den, Uiterwijk, J. W. H. M., and Rijswijk, J. van (2002). Games Solved: Now and in the Future. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 277–311. [42, 48, 111]

Heule, M. J. H. and Rothkrantz, L. J. M. (2007). Solving Games: Dependence of Applicable Solving Procedures. *Science of Computer Programming*, Vol. 67, No. 1, pp. 105–124. [42]

Hsieh, M. Y. and Tsai, S.-C. (2007). On the Fairness and Complexity of Generalized k-in-a-row Games. *Theoretical Computer Science*, Vol. 385, Nos. 1–3, pp. 88–100. [2]

Hsu, F.-H. (2002). *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, Princeton, NY, USA. [3]

Huizinga, J. (1955). *Homo Ludens; A Study of the Play-Element in Culture*. Beacon Press, Boston, MA, USA. [1]

Hyatt, R. M., Grover, A. E., and Nelson, H. L. (1990). Cray Blitz. *Computers, Chess and Cognition* (eds. T. A. Marsland and J. Schaeffer), pp. 111–130, Springer-Verlag, New York, NY, USA. [17]

Iida, H., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Herschberg, I. S. (1994). Thoughts on the Application of Opponent-Model Search. *Advances in Computer Chess 7* (eds. H. J. van den Herik, I. S. Herschberg, and J. W. H. M. Uiterwijk), pp. 61–78, Rijksuniversiteit Limburg, Maastricht, The Netherlands. [4, 64]

Irving, G., Donkers, H. H. L. M., and Uiterwijk, J. W. H. M. (2000). Solving Kalah. *ICGA Journal*, Vol. 23, No. 3, pp. 139–147. [42]

Iwata, S. and Kasai, T. (1994). The Othello Game on an n×n Board is PSPACE-complete. *Theoretical Computer Science*, Vol. 123, No. 2, pp. 329–340. [2]

Jansen, P. (1992). *Using Knowledge about the Opponent in Game-Tree Search*. Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. [4, 64]

Johnson, D. S. (1990). A Catalog of Complexity Classes. *Handbook of Theoretical Computer Science* (ed. J. van Leeuwen), Vol. A: Algorithms and Complexity, pp. 67–161. MIT Press, Cambridge, MA, USA. [28]

Jug, S. and Schadd, M. P. D. (2009). The 3^{rd} Computer Stratego World Championship. *ICGA Journal*, Vol. 32, No. 4, pp. 233–234. [76]

Julien, D. (2008). Pocket PC Jawbreaker Game. PDA Game Guide.com. http://www.pdagameguide.com/jawbreaker-game.html. [27]

Junghanns, A. (1999). *Pushing the Limits: New Developments in Single Agent Search*. Ph.D. thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada. [29]

Kasai, T., Adachi, A., and Iwata, S. (1979). Classes of Pebble Games and Complete Problems. *SIAM Journal on Computing*, Vol. 8, No. 4, pp. 574–586. [2]

Kendall, G., Parkes, A. J., and Spoerer, K. (2008). A Survey of NP-Complete Puzzles. *ICGA Journal*, Vol. 31, No. 1, pp. 13–34. [25, 26]

Kishimoto, A. and Müller, M. (2004). A General Solution to the Graph History Interaction Problem. *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04)* (eds. D. L. McGuinness and G. Ferguson), pp. 644–649, AAAI Press, Menlo Park, CA, USA. [60]

Kloetzer, J., Iida, H., and Bouzy, B. (2009). Playing Amazons Endgames. *ICGA Journal*, Vol. 32, No. 3, pp. 140–148. [6, 108]

Knuth, D. E. (1973). *The Art of Computer Programming. Volume 3: Sorting and Searching*. Addison-Wesley Publishing Company, Reading, MA, USA. [17]

Knuth, D. E. and Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326. [5, 8, 12, 13, 14, 18, 19, 21, 63, 89, 93]

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo Planning. *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *Lecture Notes in Computer Science (LNCS)*, pp. 282–293, Springer-Verlag, Berlin Heidelberg, Germany. [6, 19, 26, 30, 108]

Kocsis, L., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2001). Move Ordering using Neural Networks. *Engineering of Intelligent Systems* (eds. L. Montosori, J. Váncza, and M. Ali), Vol. 2070 of *Lecture Notes in Artificial Intelligence*, pp. 45–50, Springer-Verlag, Berlin, Germany. [15]

Korf, R. E. (1985). Depth-First Iterative Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, Vol. 27, No. 1, pp. 97–109. [3, 25, 29]

Korf, R. E. (1991). Multi-Player Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 48, No. 1, pp. 99–111. [92]

Korf, R. E. (1997). Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pp. 700–705, AAAI Press, Menlo Park, CA, USA. [3]

Kurzen, L. (2009). Reasoning about Cooperation, Actions and Preferences. *Synthese*, Vol. 169, No. 2, pp. 223–240. [90]

Lake, R., Schaeffer, J., and Lu, P. (1994). Solving Large Retrograde-Analysis Problems using a Network of Workstations. *Advances in Computer Chess 7* (eds. H. J. van den Herik, I. S. Herschberg, and J. W. H. M. Uiterwijk), pp. 135–162, Rijksuniversiteit Limburg, Maastricht, The Netherlands. [50, 51]

Lee, C.-S., Müller, M., and Teytaud, O. (2010). Special Issue on Monte Carlo Techniques and Computer Go. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 225–228. [25]

Levy, D. N. L. and Newborn, M. M. (1991). *How Computers Play Chess*. Computer Science Press, Inc., New York, NY, USA. [3]

Lidén, L. (2004). Artificial Stupidity: The Art of Intentional Mistakes. *AI Game Programming Wisdom* (ed. S. Rabin), Vol. 2, pp. 41–48, Charles River Media, Inc., Brookline, MA, USA. [5]

Lorentz, R. J. (2008). Amazons Discover Monte-Carlo. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 13–24, Springer-Verlag, Berlin, Germany. [6, 108]

Lorenz, K. and Lorenzen, P. (1978). *Dialogische Logik*. Wissenschaftliche Buchgesellschaft, Darmstadt, Germany. In German. [2]

Lorenz, U. and Tscheuschner, T. (2006). Player Modeling, Search Algorithms and Strategies in Multi Player Games. *Advances in Computer Games (ACG 2005)* (eds. H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers), Vol. 4250 of *Lecture Notes in Computer Science (LNCS)*, pp. 210–224, Springer-Verlag, Berlin, Germany. [91, 92, 108]

Luckhardt, C. A. and Irani, K. B. (1986). An Algorithmic Solution of N-Person Games. *Proceedings of the 5^{th} National Conference on Artificial Intelligence (AAAI-86)*, pp. 158–162, AAAI Press, Menlo Park, CA, USA. [8, 89, 91, 92]

Lu, H. and Xia, Z. (2008). AWT: Aspiration with Timer Search Algorithm in Siguo. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 264–274, Springer-Verlag, Berlin, Germany. [4, 76]

Marsland, T. A. (1986). A Review of Game Tree Pruning. *ICCA Journal*, Vol. 9, No. 1, pp. 3–19. [6, 11, 15]

Marsland, T. A. and Björnsson, Y. (2001). Variable-Depth Search. *Advances in Computer Games (ACG 9)* (eds. H. J. van den Herik and B. Monien), pp. 9–24, Universiteit Maastricht, Maastricht, The Netherlands. [5, 8, 69, 108, 114]

Marsland, T. A. and Popowich, F. (1985). Parallel Game-Tree Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI–7, No. 4, pp. 442–452. [14, 103]

Milton Bradley Co. (1986). *Stratego Instructions*. Springfield, MA, USA. Obtained at http://safemanuals.com/. [74]

Méhat, J. and Cazenave, T. (2010). Combining UCT and Nested Monte-Carlo Search for Single-Player General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 271–277. [40]

Mesmay, F., Rimmel, A., Voronenko, Y., and Püschel, M. (2009). Bandit-Based Optimization on Graphs with Application to Library Performance Tuning. *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)* (eds. A. P. Danyluk, L. Bottou, and M. L. Littman), pp. 729–736, ACM, New York, NY, USA. [19, 26]

Michie, D. (1966). Game-Playing and Game-Learning Automata. *Advances in Programming and Non-Numerical Computation* (ed. L. Fox), pp. 183–200, Pergamon, New York, NY, USA. [5, 8, 64]

Millington, I. (2006). *Artificial Intelligence for Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [5]

Montgomery, W. (1886). The Malagasy Game of Fanorona. *The Antananarivo Annual and Madagascar Magazine*, Vol. 10, pp. 148–156. [43]

Moribe, K. (1985). Chain Shot! *Gekkan ASCII*, No. November issue. In Japanese. [26]

Mühlenbein, H. (1997). The Equation for Response to Selection and its Use for Prediction. *Evolutionary Computation*, Vol. 5, No. 3, pp. 303–346. [33]

Myerson, R. B. (1997). *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA, USA. [2]

Myerson, R. B. (1999). Nash Equilibrium and the History of Economic Theory. *Journal of Economic Literature*, Vol. 37, No. 3, pp. 1067–1082. [2]

Nagai, A. (2002). *Df-pn Algorithm for Searching AND/OR Trees and its Applications*. Ph.D. thesis, Department of Information Science, University of Tokyo, Tokyo, Japan. [8]

Nareyek, A. (2000). Intelligent Agents for Computer Games. *Computers and Games, CG 2000* (eds. T. Marsland and I. Frank), Vol. 2063 of *Lecture Notes in Computer Science (LNCS)*, pp. 414–422, Springer-Verlag, Berlin, Germany. [5]

Nash, J. (1952). Some Games and Machines for Playing Them. Technical Report D–1164, Rand Corporation, Santa Monica, CA, USA. [42]

Neumann, J. von (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, Vol. 100, No. 1, pp. 295–320. In German. Translated by S. Bargmann (1959) as On the Theory of Games of Strategy. In *Contributions to the Theory of Games 4*, (eds. A. W. Tucker and R. D. Luce), Vol. 40 of *Annals of Mathematical Studies*, pp. 13–42, Princeton University Press, Princeton, NY, USA. [3, 5]

Neumann, J. von and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NY, USA. [2, 3, 12]

Newell, A., Shaw, C., and Simon, H. (1958). Chess Playing Programs and the Problem of Complexity. *IBM Journal of Research and Development*, Vol. 4, No. 2, pp. 320–335. Reprinted (1963) in Computers and Thought (eds. E. A. Feigenbaum and J. Feldman), pp. 39–70. McGraw-Hill, New York, NY, USA. [3]

Nijssen, J. A. M. and Winands, M. H. M. (2011). Enhancements for Multi-Player Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H. J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *Lecture Notes in Computer Science (LNCS)*, pp. 238–249, Springer-Verlag, Berlin, Germany. [108]

Nowakowski, R. J. (2009). The History of Combinatorial Game Theory. *Proceedings of Board Game Studies Colloquium XI* (ed. J. N. Silva), pp. 133–145, Associação Ludus, Lisbon, Portugal. [2]

Osborne, J. A. (2003). Markov Chains for the RISK Board Game Revisited. *Mathematics Magazine*, Vol. 2, No. 76, pp. 129–135. [64]

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press, Cambridge, MA, USA. [90]

Palay, A. J. (1983). *Searching with Probabilities*. Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. [17, 60]

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA, USA. [2]

Patashnik, O. (1980). Qubic: 4×4×4 Tic-Tac-Toe. *Mathematics Magazine*, Vol. 53, No. 4, pp. 202–216. [42]

Pauly, M. (2002). A Modal Logic for Coalitional Power in Games. *Journal of Logic and Computation*, Vol. 1, No. 12, pp. 149–166. [90]

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA, USA. [12]

Peters, H. J. M. (2008). *Game Theory: A Multi-Leveled Approach*. Springer-Verlag, Berlin, Germany. [2]

Peterson, G. L., Reif, J. H., and Azhar, S. (2002). Decision Algorithms for Multiplayer Noncooperative Games of Incomplete Information. *Computers & Mathematics with Applications*, Vol. 1–2, No. 43, pp. 179–206. [91]

Plaat, A. (1996). *Research Re: Search & Re-Search*. Ph.D. thesis, Tinbergen Institute and Department of Computer Science, Erasmus University Rotterdam, Rotterdam, The Netherlands. [53, 103]

Reif, J. H. (1984). The Complexity of Two-Player Games of Incomplete Information. *Journal of Computer and System Sciences*, Vol. 29, No. 2, pp. 274–301. [64]

Reisch, S. (1980). Gobang ist PSPACE-vollständig. *Acta Informatica*, Vol. 13, No. 1, pp. 59–66. In German. [2]

Reisch, S. (1981). Hex ist PSPACE-vollständig. *Acta Informatica*, Vol. 15, No. 2, pp. 167–191. In German. [2]

Robson, J. M. (1983). The Complexity of Go. *Proceedings of IFIP 9th World Computer Congress* (ed. R. E. A. Mason), Vol. 83 of *Information Processing*, pp. 413–417, Elsevier Science Publishers, Amsterdam, The Netherlands. [2]

Robson, J. M. (1984). N by N Checkers is Exptime Complete. *SIAM Journal on Computing*, Vol. 13, No. 2, pp. 252–267. [2]

Romein, J. W. and Bal, H. E. (2003). Solving Awari with Parallel Retrograde Analysis. *IEEE Computer*, Vol. 36, No. 10, pp. 26–33. [42, 43, 50]

Rubinstein, R. Y. (2003). The Cross-Entropy Method for Combinatorial and Continuous Optimization. *Methodology and Computing in Applied Probability*, Vol. 1, No. 2, pp. 127–190. [4, 33]

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, USA, 2nd edition. [6, 11, 63, 64]

Sackson, S. (1969). *A Gamut of Games*. Random House, New York, NY, USA. [98]

Sadikov, A. and Bratko, I. (2007). Solving 20 × 20 Puzzles. *Proceedings of the Computer Games Workshop 2007 (CGW 2007)* (eds. H. J. van den Herik, J. W. H. M. Uiterwijk, M. H. M. Winands, and M. P. D. Schadd), pp. 157–164, Maastricht University, Maastricht, The Netherlands. [29]

Saito, J.-T. and Winands, M. H. M. (2010). Paranoid Proof-Number Search. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010)* (eds. G. N. Yannakakis and J. Togelius), pp. 203–210. IEEE press, Piscataway, NJ, USA. [8, 89, 93]

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, Vol. 3, No. 3, pp. 210–229. Reprinted in (1963) *Computers and Thought* (eds. E. A. Feigenbaum and J. Feldmann), pp. 71–105. McGraw-Hill Book Company, New York, NY, USA. [12]

Satz, I. (2008). The 1st Computer Stratego World Championship. *ICGA Journal*, Vol. 31, No. 1, pp. 50–51. [75, 76]

Sauro, L., Gerbrandy, J., Hoek, W. van der, and Wooldridge, M. (2006). Reasoning about Action and Cooperation. *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-06)* (eds. P. Stone and G. Weiss), pp. 185–192, ACM Press, New York, NY, USA. [90]

Schadd, M. P. D. (2006). Solving Fanorona. M.Sc. thesis, Maastricht University, Maastricht, The Netherlands. [46]

Schadd, M. P. D. and Satz, I. (2008). The 2^{nd} Computer Stratego World Championship. *ICGA Journal*, Vol. 31, No. 4, pp. 251–252. [76]

Schadd, M. P. D. and Winands, M. H. M. (2009). Quiescence Search for Stratego. *Proceedings of the 21st BeNeLux Conference on Artificial Intelligence (BNAIC'09)* (eds. T. Calders, K. Tuyls, and M. Pechenizkiy), pp. 225–232, Technische Universiteit Eindhoven, Eindhoven, The Netherlands. [75, 78]

Schadd, M. P. D. and Winands, M. H. M. (2011). Best Reply Search for Multiplayer Games. *Transactions on Computational Intelligence and AI in Games*, Vol. 3, No. 1, pp. 57–66. [89]

Schadd, M. P. D., Winands, M. H. M., Bergsma, M. H. J., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2007a). Fanorona is a Draw. *ICGA Journal*, Vol. 30, No. 1, pp. 43–45. [41]

Schadd, M. P. D., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bergsma, M. H. J. (2007b). Best Play in Fanorona Leads to Draw. *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)* (ed. P. Wang *et al.*), pp. 635–642, World Scientific Publishing Co. Pte. Ltd., Singapore. [41]

Schadd, M. P. D., Winands, M. H. M., Chaslot, G. M. J.-B., Herik, H. J. van den, and Uiterwijk, J. W. H. M. (2008a). Single-Player Monte-Carlo Tree Search. *Proceedings of the 20st BeNeLux Conference on Artificial Intelligence (BNAIC'08)* (eds. A. Nijholt, M. Pantic, M. Poel, and H. Hondorp), pp. 361–362, University of Twente, Enschede, The Netherlands. [25]

Schadd, M. P. D., Winands, M. H. M., Herik, H. J. van den, and Aldewereld, H. (2008b). Addressing NP-Complete Puzzles with Monte-Carlo Methods. *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning*, Vol. 9, pp. 55–61, The Society for the Study of Artificial Intelligence and Simulation of Behaviour, Brighton, United Kingdom. [25]

Schadd, M. P. D., Winands, M. H. M., Herik, H. J. van den, Chaslot, G. M. J.-B., and Uiterwijk, J. W. H. M. (2008c). Single-Player Monte-Carlo Tree Search. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 1–12, Springer-Verlag, Berlin, Germany. [25, 26, 28, 31, 38]

Schadd, M. P. D., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bergsma, M. H. J. (2008d). Best Play in Fanorona Leads to Draw. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 369–387. [41]

Schadd, M. P. D., Winands, M. H. M., and Uiterwijk, J. W. H. M. (2009). ChanceProbCut: Forward Pruning in Chance Nodes. *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG 2009)* (ed. P. L. Lanzi), pp. 178–185, IEEE press, Piscataway, NJ, USA. [63]

Schaeffer, J. (1983). The History Heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19. [16, 77, 101]

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer-Verlag, New York, NY, USA. [8, 50, 60]

Schaeffer, J. (2007). Game Over: Black to Play and Draw in Checkers. *ICGA Journal*, Vol. 30, No. 4, pp. 187–197. [42, 60]

Schaeffer, J. and Plaat, A. (1996). New Advances in Alpha–Beta Searching. *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*, pp. 124–130, ACM Press, New York, NY, USA. [18, 76]

Schaeffer, J., Björnsson, Y., Burch, N., Lake, R., Lu, P., and Sutphen, S. (2003). Building the Checkers 10-Piece Endgame Databases. *Advances in Computer Games 10* (eds. H. J. van den Herik, H. Iida, and E. A. Heinz), pp. 193–210, Kluwer Academic Publishers, Dordrecht, The Netherlands. [50]

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is Solved. *Science*, Vol. 317, No. 5844, pp. 1518–1522. [3, 7, 8, 42, 47, 60]

Scott, B. (2002). The Illusion of Intelligence. *AI Game Programming Wisdom* (ed. S. Rabin), Vol. 1, pp. 16–20, Charles River Media, Inc., Brookline, MA, USA. [5]

Sevenster, M. (2006). *Branches of Imperfect Information: Logic Games and Computation*. Ph.D. thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, Amsterdam, The Netherlands. [2, 4, 63]

Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, Vol. 41, No. 7, pp. 256–257. [3]

Sheppard, B. (2002a). World-Championship-Caliber Scrabble. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 241–275. [3]

Sheppard, B. (2002b). *Towards Perfect Play of Scrabble*. Ph.D. thesis, Department of Computer Science, Maastricht University, Maastricht, The Netherlands. [3, 18]

Skowronski, P., Björnsson, Y., and Winands, M. H. M. (2010). Automated Discovery of Search-Extension Features. *Advances in Computer Games (ACG 2009)* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science (LNCS)*, pp. 182–194, Springer-Verlag, Berlin, Germany. [76]

Slate, J. D. and Atkin, L. R. (1977). CHESS 4.5: The Northwestern University Chess Program. *Chess Skill in Man and Machine* (ed. P. W. Frey), pp. 82–118, Springer-Verlag, New York, NY, USA. [17, 77]

Smith, S. J. J. and Nau, D. S. (1993). Toward an Analysis of Forward Pruning. Technical Report CS-TR-3096, University of Maryland at College Park, College Park, MD, USA. [70]

Smith, S. J. J., Nau, D., and Throop, T. (1998). Computer Bridge: A Big Win for AI Planning. *AI Magazine*, Vol. 19, No. 2, pp. 93–105. [64]

Spronck, P. (2005). *Adaptive Game AI.* Ph.D. thesis, Department of Computer Science, Maastricht University, Maastricht, The Netherlands. [5]

Stankiewicz, J. A. and Schadd, M. P. D. (2009). Opponent Modeling in Stratego. *Proceedings of the 21st BeNeLux Conference on Artificial Intelligence (BNAIC'09)* (eds. T. Calders, K. Tuyls, and M. Pechenizkiy), pp. 233–240, Technische Universiteit Eindhoven, Eindhoven, The Netherlands. [64]

Stengård, K. (2006). Utveckling av Minimax-Baserad Agent för Strategispelet Stratego. M.Sc. thesis, Lund University, Sweden. In Swedish. [66, 75]

Stockman, G. C. (1979). A Minimax Algorithm Better than Alpha-Beta? *Artificial Intelligence*, Vol. 12, No. 2, pp. 179–196. [6]

Ströhlein, T. (1970). *Untersuchungen über Kombinatorische Spiele.* Ph.D. thesis, Fakultät für Allgemeine Wissenschaften, Technischen Hochschule München, München, Germany. In German. [8, 43, 48, 50]

Sturtevant, N. R. (2003a). *Multi-Player Games: Algorithms and Approaches.* Ph.D. thesis, Computer Science Department, University of California, Los Angeles, CA, USA. [4, 92, 93, 97, 101, 108, 114]

Sturtevant, N. R. (2003b). Last-Branch and Speculative Pruning Algorithms for Max^n. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)* (eds. G. Gottlob and T. Walsh), pp. 669–678, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [8, 89, 92, 101]

Sturtevant, N. R. (2003c). A Comparison of Algorithms for Multi-Player Games. *Computers and Games (CG 2002)* (eds. J. Schaeffer, M. Müller, and Y. Björnsson), Vol. 2883 of *Lecture Notes in Computer Science (LNCS)*, pp. 108–122, Springer-Verlag, Berlin, Germany. [89, 92, 93, 95, 101, 102]

Sturtevant, N. R. (2008a). An Analysis of UCT in Multi-Player Games. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 37–49, Springer-Verlag, Berlin, Germany. [4, 25, 97, 108, 114]

Sturtevant, N. R. (2008b). An Analysis of UCT in Multi-Player Games. *ICGA Journal*, Vol. 31, No. 4, pp. 195–208. [22, 97]

Sturtevant, N. R. and Bowling, M. H. (2006). Robust Game Play Against Unknown Opponents. *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)* (eds. H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone), pp. 713–719, ACM, Hakodate, Japan. [92]

Sturtevant, N. R. and Korf, R. (2000). On Pruning Techniques for Multi-Player Games. *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00)*, pp. 201–207, AAAI Press, Menlo Park, CA, USA. [4, 5, 8, 89, 91, 92, 93]

Sturtevant, N. R., Zinkevich, M., and Bowling, M. H. (2006). Prob-Max^n: Playing N-player Games with Opponent Models. *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pp. 1057–1063, AAAI Press, Menlo Park, CA, USA. [92]

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, Vol. 3, No. 1, pp. 9–44. [3]

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA. [31, 33]

Tak, M. J. W. (2010). The Cross-Entropy Method Applied to SameGame. Bachelor thesis, Maastricht University, Maastricht, The Netherlands. [31, 34, 37, 38]

Takes, F. W. and Kosters, W. A. (2009). Solving SameGame and its Chessboard Variant. *Proceedings of the 21st Benelux Conference on Artificial Intelligence (BNAIC'09)* (eds. T. Calders, K. Tuyls, and M. Pechenizkiy), pp. 249–256, Technische Universiteit Eindhoven, Eindhoven, The Netherlands. [28, 29, 31]

Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, Vol. 6, No. 2, pp. 215–219. [3]

Tesauro, G. and Galperin, G. R. (1997). On-line Policy Improvement using Monte-Carlo Search. *Advances in Neural Information Processing Systems* (eds. M. C. Mozer, M. I. Jordan, and T. Petsche), Vol. 9, pp. 1068–1074, MIT Press, Cambridge, MA, USA. [18]

Teytaud, F. and Teytaud, O. (2010). Creating an Upper-Confidence-Tree Program for Havannah. *Advances in Computer Games (ACG 2009)* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science (LNCS)*, pp. 65–74, Springer-Verlag, Berlin, Germany. [22]

Thiery, C. and Scherrer, B. (2009). Improvements on Learning Tetris with Cross Entropy. *ICGA Journal*, Vol. 32, No. 1, pp. 23–33. [4]

Thomson, T. (2000). Lambda-Search in Game Trees - With Application to Go. *ICGA Journal*, Vol. 23, No. 4, pp. 203–217. [43]

Tozour, P. (2002). The Perils of AI Scripting. *AI Game Programming Wisdom* (ed. S. Rabin), Vol. 1, pp. 541–547, Charles River Media, Inc., Brookline, MA, USA. [5]

Treijtel, C. and Rothkrantz, L. J. M. (2001). Stratego Expert System Shell. *Proceedings of the 2nd International Conference on Intelligent Games and Simulation (GAME-ON 2001)* (eds. Q. H. Mehdi, N. E. Gough, and D. Al-Dabbas), pp. 17–21, The European Multidisciplinary Society for Modelling and Simulation Technology (Eurosis), Ostend, Belgium. [75]

Tromp, J. T. (2008). Solving Connect-4 on Medium Board Sizes. *ICGA Journal*, Vol. 31, No. 2, pp. 110–112. [42]

Turing, A. M. (1953). Digital Computers Applied to Games. *Faster than Thought* (ed. B. V. Bowden), pp. 286–297, Pitman, London, United Kingdom. [3]

Uiterwijk, J. W. H. M. (1992). The Countermove Heuristic. *ICCA Journal*, Vol. 15, No. 1, pp. 8–15. [16]

Vempaty, N. R., Kumar, V., and Korf, R. E. (1991). Depth-First versus Best-First Search. *Proceedings of the 9^{th} National Conference on Artificial Intelligence (AAAI-91)*, pp. 434–440, AAAI Press, Menlo Park, CA, USA. [30]

Veness, J. and Blair, A. (2007). Effective Use of Transposition Tables in Stochastic Game Tree Search. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games (CIG 2007)* (eds. A. Blair, S-B. Cho, and S. M. Lucas), pp. 112–116, IEEE press, Piscataway, NJ, USA. [70, 76, 77]

Wágner, J. and Virág, I. (2001). Solving Renju. *ICGA Journal*, Vol. 24, No. 1, pp. 30–34. [42]

Winands, M. H. M. (2004). *Informed Search in Complex Games*. Ph.D. thesis, Department of Computer Science, Maastricht University, Maastricht, The Netherlands. [60]

Winands, M. H. M. (2007). SIA Wins Surakarta Tournament. *ICGA Journal*, Vol. 30, No. 3, p. 162. [60]

Winands, M. H. M. (2008). 6×6 LOA is Solved. *ICGA Journal*, Vol. 31, No. 4, pp. 234–238. [58]

Winands, M. H. M. and Björnsson, Y. (2010). Evaluation Function based Monte-Carlo LOA. *Advances in Computer Games (ACG 2009)* (eds. H. J. van den Herik and P. Spronck), Vol. 6048 of *Lecture Notes in Computer Science (LNCS)*, pp. 33–44, Springer-Verlag, Berlin, Germany. [6, 108]

Winands, M. H. M. and Schadd, M. P. D. (2011). Evaluation-Function Based Proof-Number Search. *Computers and Games (CG 2010)* (eds. H. J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *Lecture Notes in Computer Science (LNCS)*, pp. 23–35, Springer-Verlag, Berlin, Germany. [61, 113]

Winands, M. H. M., Herik, H. J. van den, Uiterwijk, J. W. H. M., and Werf, E. C. D. van der (2005). Enhanced Forward Pruning. *Information Sciences*, Vol. 175, No. 4, pp. 315–329. [69]

Winands, M. H. M., Werf, E. C. D. van der, Herik, H. J. van den, and Uiterwijk, J. W. H. M. (2006). The Relative History Heuristic. *Computers and Games (CG 2004)* (eds. H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu), Vol. 3846 of *Lecture Notes in Computer Science (LNCS)*, pp. 262–272, Springer-Verlag, Berlin, Germany. [16]

Winands, M. H. M., Björnsson, Y., and Saito, J.-T. (2008). Monte-Carlo Tree Search Solver. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *Lecture Notes in Computer Science (LNCS)*, pp. 25–36, Springer-Verlag, Berlin, Germany. [21]

Xia, Z. Y., Hu, Y. A., Wang, J., Jiang, Y.-C., and Qin, X. L. (2005). Analyze and Guess Type of Piece in the Computer Game Intelligent System. *Fuzzy Systems and Knowledge Discovery, Second International Conference (FSKD 2005)* (eds. L. Wang and Y. Jin), Vol. 3614 of *Lecture Notes in Computer Science, (LNCS/LNAI)*, pp. 1174–1183, Springer-Verlag, Berlin Heidelberg, Germany. [4, 76]

Xia, Z. Y., Zhu, Y., and Lu, H. (2007). Using the Loopy Belief Propagation in Siguo. *ICGA Journal*, Vol. 30, No. 4, pp. 209–220. [4, 76]

Zermelo, E. (1913). Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. *Proceedings of the Fifth Congress of Mathematicians, Cambridge 1912*, pp. 501–504, Cambridge University Press, Cambridge, United Kingdom. In German. [2]

Zobrist, A. L. (1970). A New Hashing Method with Application for Game Playing. Technical Report 88, Computer Science Department, The University of Wisconsin, Madison, WI, USA. Reprinted in (1990) *ICCA Journal*, Vol. 13, No. 2, pp. 69–73. [17]

Zuckerman, I., Felner, A., and Kraus, S. (2009). Mixing Search Strategies for Multi-Player Games. *Proceedings of the Twenty-first International Joint Conferences on Artificial Intelligence (IJCAI-09)* (ed. C. Boutilier), pp. 646–651, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. [8, 89, 91, 108, 114]

Index

Summary

This thesis investigates how selective-search methods can improve the performance of a game program for a given domain. Selective-search methods aim to explore only the profitable parts of the state space, but take the risk to overlook the best move. We propose several selective-search methods and test them in a large number of game domains.

In Chapter 1 we provide a brief introduction on games research and discuss selective search for games. The following problem statement guides our research.

> **Problem statement**: *How can we improve selective-search methods in such a way that programs increase their performance in domains of different complexity?*

Rather than testing selective-search methods on one class of games, we chose different classes of games, which all have to be addressed differently. Each class of games represents a level of complexity. Between every level there exists a *complexity jump*. With a complexity jump the complexity of the game increases significantly because the mechanism of the game is changed. We have chosen five different levels of games, resulting in four complexity jumps. (1) One-player games, or puzzles, involve no opponent and are a testbed for planning algorithms. (2) Two-player games are the classic testbed for search methods. We use them for investigating mate-solvers. For testing search with chance nodes, (3) non-deterministic and (4) imperfect-information games may be used. (5) Multi-player games are a testbed for dealing with coalition forming. We formulate four research questions to guide our research. Each one deals with search for a different class of games and a different selective-search method. The four research questions address (1) Monte-Carlo Tree Search (MCTS), (2) Proof-Number (PN) search, (3) expectimax, and (4) multi-player search.

Chapter 2 is a general introduction to search methods for games. It explains the minimax algorithm and the well-known $\alpha\beta$ search. Standard techniques for enhancing the $\alpha\beta$ search are discussed as well. We furthermore explain MCTS and its enhancements.

The traditional approach to deterministic one-player games with perfect information is applying A* or IDA*. These methods have been quite successful in coping

with this class of games. The disadvantage of these methods is that they require an admissible heuristic evaluation function. The construction of such a function can be difficult. Since the selective-search method MCTS does not require an admissible heuristic, it may be an interesting alternative. This has led us to the first research question.

> **Research question 1**: *How can we adapt Monte-Carlo Tree Search for a one-player game?*

Chapter 3 answers the first research question by proposing a new MCTS variant called Single-Player Monte-Carlo Tree Search (SP-MCTS). MCTS is adapted by two modifications resulting in SP-MCTS. The modifications concern (1) the selection strategy and (2) the backpropagation strategy. The selection strategy is adapted to take the standard deviation of the scores for a move into account and to combine the average score with the top score. The backpropagation strategy is modified to provide the selection strategy with all required information. For evaluating SP-MCTS, the puzzle SameGame is chosen as test domain. So far, there does not exist a good admissible heuristic evaluation function for this game.

On the standardized test set of 20 SameGame positions, the manually tuned SP-MCTS method, which invests all search time at the initial position, scored 73,998 points. This was the highest score on the test set at that point of time (2008). The main contribution is therefore that we successfully adapted MCTS for a one-player game. Inspired by our approach, two other Monte-Carlo-based approaches, Nested Monte-Carlo and Heuristically Guided Swarm Tree Search, broke our record subsequently. At the time of publishing this thesis SP-MCTS, with parameters tuned by the Cross-Entropy Method and with time equally distributed over the consecutive positions, scored 78,012 points on the test set, which is currently the third highest score (2010). Thus, answering research question 1, we have shown that MCTS is applicable to a one-player deterministic perfect-information game. Our variant, SP-MCTS, is able to achieve good results in the game of SameGame. SP-MCTS is a worthy alternative for puzzles where a good admissible estimator cannot be found.

Ideally, a search method is able to prove that a move is the optimal one for a given game. The game is solved if this is achieved. A selective-search method specially designed for solving (end)games is Proof-Number (PN) search. PN search is efficient in searching game trees with a non-uniform branching factor. Moreover, for quite some games, endgame databases played a role in solving. This has led us to the second research question.

> **Research question 2**: *How can we solve a two-player game by using Proof-Number search in combination with endgame databases?*

Chapter 4 answers the second research question by examining the tradeoff between time spent on PN search and time spent on creating endgame databases when solving the game of Fanorona. This game has a material-based theme with a state-space complexity similar to checkers. Endgame-database statistics show that (1) the player to move has an advantage and (2) that a draw can often be achieved in

spite of having fewer pieces than the opponent. The optimal endgame-database sizes for the 3×9, 5×5, and 7×5 Fanorona variants are 3, 4, and 5 pieces, respectively. We conclude that the optimal database size is located at the point where the time required for database construction and the time required for solving by PN search are of the same order. Our main result is that standard Fanorona (5×9) has been weakly solved and is drawn when both players play optimally, adding Fanorona to the list of solved games. This result was achieved by combining the PN-search variant PN^2 with all endgame databases up to 7 pieces.

Another form of searching selectively in two-player deterministic games with perfect information is variable-depth search. Branches can be pruned if they seem unpromising (forward pruning), or extended if the branches are promising (search extensions). There exist several successful forward-pruning techniques for the $\alpha\beta$ algorithm. For two-player games with non-determinism or imperfect information expectimax may be used. Expectimax adds chance nodes to the search tree. There are, however, no forward-pruning techniques available for chance nodes. This has led us to the third research question.

> **Research question 3**: *How can we perform forward pruning at chance nodes in the expectimax framework?*

Chapter 5 answers the third research question by proposing the forward-pruning technique ChanceProbCut for expectimax. This technique is the first in its kind to forward prune at chance nodes. ChanceProbCut is inspired by the $\alpha\beta$ forward-pruning technique ProbCut. ChanceProbCut estimates values of chance events based on shallow searches. Based on the correlation between evaluations obtained from searches at different depths, ChanceProbCut prunes chance events in advance if the result of the chance node probably falls outside the search window. Two non-deterministic games (Dice and ChanceBreakthrough) and a game of imperfect information (Stratego) served as test domains. Experiments revealed that ChanceProbCut is able to reduce the size of the game tree significantly without a loss of decision quality in Stratego, Dice, and ChanceBreakthrough. A safe node reduction of between 30% and 85% is achieved across all games. Thus, ChanceProbCut finds the best move faster in the expectimax framework, while not affecting the playing strength. The gained time may be invested in a deeper search. Selfplay experiments in Stratego and Dice showed that there is a small but relevant improvement in playing strength. In ChanceBreakthrough, though, a significant increase in performance was measured. ChanceProbCut is able to win 54.4% on 4,000 games.

In deterministic two-player games with perfect information, the majority of research focused on the $\alpha\beta$ algorithm. For deterministic multi-player games with perfect information, the choice of algorithm is not as straightforward. The two main algorithms are called $\max^n$ and paranoid, both approaching the problem from a different angle. Max^n assumes that every player tries to maximize the own score, while paranoid assumes that all opponents form a coalition against the root player. However, these assumptions have drawbacks. Due to the lack of safe pruning in $\max^n$ only a limited lookahead is possible. Furthermore, the underlying assumption

of $\max^n$ may be unrealistic, resulting in $\max^n$ to be too optimistic. When searching deep with the paranoid algorithm, the other players may dominate the root player, resulting in paranoid to be too pessimistic. This has led us to the fourth research question.

> **Research question 4**: *How can we improve search for multi-player games?*

Chapter 6 answers the fourth research question by proposing a new search algorithm, called Best-Reply Search (BRS), for deterministic non-cooperative multi-player games with perfect information. This algorithm allows only one opponent to play a counter move. This opponent is the one with the strongest move against the root player. The other players have to pass their turn. Using this approach, more turns of the root player can be searched, resulting in long-term planning. At the same time, some sort of cautiousness is preserved by searching the strongest opponent move.

We have chosen three deterministic multi-player games of perfect information, i.e., Chinese Checkers, Focus, and Rolit. BRS is able to significantly outperform $\max^n$ in these games, with a win ratio of between 65% and 95%. Against paranoid, BRS is significantly stronger in Chinese Checkers and Focus, with win ratios of between 57% and 71%. In Rolit, BRS and paranoid are on equal footing. When playing different kind of opponents at the same time, BRS is the strongest algorithm in Chinese Checkers and Focus. In Rolit, BRS is somewhat behind paranoid. Increasing the search time generally does not have a negative effect on the performance of BRS. This implies that searching illegal positions, which are generated by forcing opponents to pass, does not have a large influence. The possible negative effect is outbalanced by the larger lookahead.

Chapter 7 concludes the thesis and gives an outlook on open questions and directions for future research. Taking the answers to the research questions above into account we see that there are four ways to improve selective-search methods. First, Single-Player Monte-Carlo Tree Search balances exploitation and exploration such that it is a worthy alternative for one-player games where a good admissible estimator cannot be found. Second, PN search with endgame databases, which prefers narrow subtrees above wide ones, is able to prove that the game-theoretic value of the two-player game Fanorona is a draw. Third, ChanceProbCut can forward prune chance events in two-player games with non-determinism or imperfect information. Fourth, in non-cooperative deterministic multi-player games with perfect information, Best-Reply Search achieves long-term planning by assuming that only one opponent is allowed to play a counter move.

While this thesis shows that selective-search methods are successful in quite some domains, all our proposed enhancements and algorithms can be tested in other domains, as well. These domains include classes of games with corresponding complexity levels, but also game classes which have not been covered in this research. We mention one-player games with non-determinism, two-player games with non-determinism and imperfect information, and multi-player games with non-determinism and/or imperfect information.

Samenvatting

Dit proefschrift onderzoekt hoe selectieve zoekmethoden de prestaties van een spelprogramma kunnen verbeteren voor een bepaald domein. Selectieve zoekmethoden hebben als doel om alleen de winstgevende delen van de zoekruimte te verkennen, maar ze nemen het risico om de beste zet te overzien. Wij stellen een aantal selectieve zoekmethoden voor en testen ze in een groot aantal speldomeinen.

In hoofdstuk 1 geven we een korte inleiding in onderzoek in spelen en bespreken selectieve zoekmethoden voor spelen. De volgende probleemstelling stuurt ons onderzoek.

> **Probleemstelling**: *Hoe kunnen we selectieve zoekmethoden op een zodanige wijze verbeteren dat programma's hun prestaties in domeinen van verschillende complexiteit verhogen?*

In plaats van selectieve zoekmethoden te testen op één klasse van spelen, kozen we verschillende klassen van spelen, die allemaal anders aangepakt moeten worden. Elke spelklasse vertegenwoordigt een niveau van complexiteit. Tussen elk niveau bestaat er een *complexiteitssprong*. Met een complexiteitssprong wordt de complexiteit van het spel aanzienlijk verhoogd, omdat het mechanisme van het spel is veranderd. We hebben gekozen voor vijf verschillende niveaus van spelen, wat resulteert in vier complexiteitssprongen. (1) Éénspeler spelen, of puzzels, hebben geen tegenstander en zijn een testdomein voor planningalgoritmes. (2) Tweespeler spelen zijn het klassieke testdomein voor zoekmethoden. We gebruiken ze voor zoekmethoden die de speltheoretische waarde kunnen bewijzen. Voor zoeken met kansknopen kunnen (3) niet-deterministische en (4) imperfecte-informatie spelen worden gebruikt. (5) Meerspeler spelen zijn een testdomein voor het omgaan met coalitievorming. We formuleren vier onderzoeksvragen die ons onderzoek sturen. Elke vraag heeft als thema het zoeken in een andere klasse van spelen en een andere selectieve zoekmethode. De vier onderzoeksvragen gaan over (1) Monte-Carlo Tree Search (MCTS), (2) Proof-Number (PN) search, (3) expectimax, en (4) meerspeler zoekmethoden.

Hoofdstuk 2 is een algemene inleiding in zoekmethoden voor spelen. Het beschrijft het minimax algoritme en de bekende $\alpha\beta$ zoekmethode. Standaardtechnieken voor het verbeteren van de $\alpha\beta$ zoekmethode worden ook besproken. Verder leggen wij MCTS en de bijbehorende verbeteringen uit.

De traditionele benadering voor deterministische éénspeler spelen met perfecte informatie is het toepassen van A* of IDA*. Deze methoden zijn zeer succesvol in het omgaan met deze klasse van spelen. Het nadeel van deze methoden is dat ze een onderschattende evaluatiefunctie vereisen. Het maken van een dergelijke functie kan moeilijk zijn. Omdat de selectieve zoekmethode MCTS geen evaluatiefunctie nodig heeft, is het een mogelijk interessant alternatief. Dit heeft geleid tot de eerste onderzoeksvraag.

> **Onderzoeksvraag 1**: *Hoe kunnen we Monte-Carlo Tree Search aanpassen voor een éénspeler spel?*

Hoofdstuk 3 geeft antwoord op de eerste onderzoeksvraag door een nieuwe MCTS variant voor te stellen, genaamd Single-Player Monte-Carlo Tree Search (SP-MCTS). Twee wijzigingen in MCTS resulteren in SP-MCTS. De wijzigingen hebben betrekking op (1) de selectiestrategie en (2) de propagatiestrategie. De selectiestrategie is aangepast door rekening te houden met de standaarddeviatie van de scores van een zet en door de gemiddelde score te combineren met de hoogste score. De propagatiestrategie is aangepast om de selectiestrategie van alle benodigde informatie te voorzien. Voor de evaluatie van SP-MCTS, is de puzzel SameGame gekozen als testdomein. Tot dusver bestaat er geen goede onderschattende heuristische evaluatiefunctie voor dit spel.

Voor 20 gestandaardiseerde SameGame testposities scoorde de handmatig ingestelde SP-MCTS methode, waarbij alle zoektijd geïnvesteerd werd aan het begin van het spel, 73.998 punten. Dit was de hoogste score op de testposities in 2008. De belangrijkste bijdrage is dan ook dat wij met succes MCTS aangepast hebben voor een éénspeler spel. Geïnspireerd door onze aanpak, hebben twee andere Monte-Carlo gebaseerde methoden, Nested Monte-Carlo Search en Heuristically Guided Swarm Tree Search, ons record later verbroken. Op het moment van publicatie van dit proefschrift heeft SP-MCTS, met parameters afgesteld door de Cross-Entropy Method en met de tijd gelijkmatig verdeeld over alle zetten, 78.012 punten gescoord op de testposities, wat momenteel de derde hoogste score is (2010). Dus, om onderzoeksvraag 1 te beantwoorden, we hebben aangetoond dat MCTS toepasbaar is op een deterministisch éénspeler spel met perfecte informatie. Onze variant, SP-MCTS, is in staat om goede resultaten te behalen in SameGame. SP-MCTS is een waardig alternatief voor puzzels waar geen goede onderschattende evaluatiefunctie gevonden kan worden.

Idealiter is een zoekmethode in staat om te bewijzen dat een zet optimaal is voor een bepaald spel. Het spel is opgelost als dit wordt bereikt. Een selectieve zoekmethode speciaal ontworpen voor het oplossen van (eind)spelen is Proof-Number (PN) search. PN search is efficiënt in het doorzoeken van spelbomen met een niet-uniforme vertakkingsgraad. Daarnaast hebben voor veel spelen eindspeldatabases een rol gespeeld bij het oplossen. Dit heeft geleid tot de tweede onderzoeksvraag.

> **Onderzoeksvraag 2**: *Hoe kunnen we een tweespeler spel oplossen met behulp van Proof-Number search in combinatie met eindspeldatabases?*

Hoofdstuk 4 geeft antwoord op de tweede onderzoeksvraag door het onderzoeken van de verhouding tussen de tijd besteed aan PN search en de tijd besteed aan het creëren van de eindspeldatabases bij het oplossen van het spel Fanorona. Dit spel is qua complexiteit van de toestandsruimte vergelijkbaar met Engels dammen. Uit statistieken van de eindspeldatabases blijkt dat (1) de speler aan zet een voordeel heeft en (2) dat ook met minder stukken op het bord dan de tegenstander een remise vaak kan worden gerealiseerd. De optimale grootten van de eindspeldatabase voor de 3×9, 5×5, en 7×5 Fanorona varianten zijn respectievelijk 3, 4, en 5 stukken. We concluderen dat de optimale grootte van de database op het punt ligt waar de tijd die nodig is om de database uit te rekenen en de tijd die nodig is voor het oplossen met PN search van dezelfde orde zijn. Ons belangrijkste resultaat is dat standaard Fanorona (5×9) is opgelost. Het resultaat is een remise als beide spelers optimaal spelen. Fanorona is dus toegevoegd aan de lijst van opgeloste spelen. Dit resultaat werd bereikt door het combineren van de PN-search variant PN^2 met alle eindspeldatabases tot 7 stukken.

Een andere vorm van selectieve zoekmethoden in deterministische tweespeler spelen met perfecte informatie is variable-depth search. Takken kunnen worden gesnoeid als ze weinig belovend lijken (voorwaarts snoeien), of verlengd indien de takken veelbelovend zijn (zoekextensies). Er bestaan een aantal succesvolle voorwaarts snoeitechnieken voor de $\alpha\beta$ zoekmethode. Voor tweespeler spelen die niet deterministisch zijn of imperfecte informatie hebben kan expectimax worden gebruikt. Expectimax voegt kansknopen toe aan de zoekboom. Er zijn echter geen voorwaarts snoeitechnieken beschikbaar voor de kansknopen. Dit heeft geleid tot de derde onderzoeksvraag.

> **Onderzoeksvraag 3**: *Hoe kunnen we voorwaarts snoeien in de kansknopen van expectimax?*

Hoofdstuk 5 geeft antwoord op de derde onderzoeksvraag door de voorwaarts snoeitechniek ChanceProbCut te beschrijven voor expectimax. Deze techniek is de eerste in zijn soort om voorwaarts te snoeien in kansknopen. ChanceProbCut is geïnspireerd door de voorwaarts snoeitechniek ProbCut voor $\alpha\beta$. ChanceProbCut schat waarden van kansgebeurtenissen gebaseerd op ondiepe zoekopdrachten. Gebaseerd op een correlatie tussen evaluaties van zoekopdrachten met verschillende dieptes, kan ChanceProbCut kansknopen voorwaarts snoeien als het resultaat van de kansknoop waarschijnlijk buiten het zoekvenster valt. Twee niet-deterministische spelen (Dice en ChanceBreakthrough) en één imperfecte-informatie spel (Stratego) dienden als testdomeinen. De experimenten laten zien dat ChanceProbCut in staat is om grote delen van de zoekboom zonder kwaliteitsverlies te snoeien. Een veilige reductie in knopen van tussen de 30% en 85% kan worden bereikt voor alle spelen. Dus ChanceProbCut vindt de beste zet sneller in expectimax zonder de speelsterkte te beïnvloeden. De uitgespaarde tijd kan worden geïnvesteerd in een diepere zoekboom. Experimenteel is gebleken dat er een kleine, maar relevante verbetering in speelsterkte is in Stratego en Dice. In ChanceBreakthrough werd een significante toename van de speelsterkte gemeten. ChanceProbCut wint 54,4% van 4.000 partijen.

In deterministische tweespeler spelen met perfecte informatie is het merendeel van het onderzoek gericht op de $\alpha\beta$ zoekmethode. Voor deterministische meerspeler spelen met perfecte informatie is de keuze van zoekmethode niet zo eenvoudig. De twee belangrijkste zoekmethoden zijn max^n en paranoid. Ze benaderen beide het probleem vanuit een andere hoek. Max^n gaat ervan uit dat elke speler probeert de eigen score te maximaliseren, terwijl paranoid ervan uitgaat dat alle tegenstanders een coalitie gesloten hebben. Deze aannames hebben echter een nadeel. Vanwege het ontbreken van veilige snoeitechnieken kan max^n slechts een beperkte zoekdiepte bereiken. Bovendien kan de onderliggende aanname van max^n onrealistisch zijn. Dit kan resulteren in een te optimistisch spel. Bij het diepe zoeken met de paranoid zoekmethode kan de coalitie van andere spelers te dominant worden. Dit kan resulteren in een te pessimistisch spel. Dit heeft geleid tot de vierde onderzoeksvraag.

> **Onderzoeksvraag 4**: *Hoe kunnen we het zoeken in meerspeler spelen verbeteren?*

Hoofdstuk 6 geeft antwoord op de vierde onderzoeksvraag door een nieuwe zoekmethode voor te stellen, genaamd Best-Reply Search (BRS), voor deterministische niet-coöperatieve meerspeler spelen met perfecte informatie. Dit algoritme laat slechts één tegenstander een zet spelen. Deze tegenstander is degene met de sterkste tegenzet. De andere spelers moeten passen. Met behulp van deze aanpak kunnen meer beurten van de wortelspeler worden doorzocht. Dit resulteert in een langetermijnplanning. Tegelijkertijd wordt een soort van voorzichtigheid bewaard door rekening te houden met de sterkste tegenzet.

We hebben gekozen voor drie deterministische meerspeler spelen met perfecte informatie, Chinese Checkers, Focus, en Rolit. BRS is in staat om vergeleken met max^n significant beter te presteren in deze spelen, en wint tussen de 65% en 95% van alle partijen. Tegen paranoid is BRS aanzienlijk sterker in Chinese Checkers en Focus, en wint tussen de 57% en 71% van alle partijen. In Rolit zijn BRS en paranoid gelijkwaardig. Bij het spelen tegen verschillende soorten tegenstanders tergelijkertijd is BRS de sterkste zoekmethode in Chinese Checkers en Focus. In Rolit is BRS enigszins zwakker dan paranoid. In het algemeen heeft het verhogen van de zoektijd geen negatief effect op de prestaties van BRS. Dit impliceert dat het zoeken van illegale posities, die worden gegenereerd door tegenstanders geforceerd te laten passen, geen groot invloed heeft. Een mogelijk negatief effect kan worden goedgemaakt door een grotere zoekdiepte.

Hoofdstuk 7 geeft de conclusies van het proefschrift en geeft een vooruitblik op open vragen en aanwijzingen voor toekomstig onderzoek. Rekening houdend met de antwoorden op de bovenstaande onderzoeksvragen zien we dat er vier manieren zijn om selectieve zoekmethoden te verbeteren. Ten eerste, Single-Player Monte-Carlo Tree Search balanceert de exploitatie en exploratie zodanig dat het een waardig alternatief voor éénspeler spelen is waar geen goede evaluatiefunctie kan worden gevonden. Ten tweede, PN search met eindspeldatabases is in staat te bewijzen dat de speltheoretische waarde van het tweespeler spel Fanorona een remise is. Ten derde, ChanceProbCut maakt het mogelijk om takken van kansknopen voorwaarts

te snoeien in tweespeler spelen met niet-determinisme of imperfecte informatie. Ten vierde, in niet-coöperatieve deterministische meerspeler spelen met perfecte informatie realiseert Best-Reply Search langetermijnplanning door te veronderstellen dat het slechts één tegenstander toegestaan is om een tegenzet te spelen.

Hoewel dit proefschrift aantoont dat selectieve zoekmethoden succesvol zijn in veel domeinen, kunnen de door ons voorgestelde verbeteringen en zoekmethoden ook in andere domeinen worden getest. Deze domeinen omvatten klassen van spelen met hetzelfde niveau van complexiteit, maar ook klassen van spelen die niet zijn opgenomen in dit onderzoek. We noemen éénspeler spelen met niet-determinisme, tweespeler spelen met niet-determinisme en imperfecte informatie, en meerspeler spelen met niet-determinisme en/of imperfecte informatie.

Curriculum Vitae

Maarten Schadd was born on June 7, 1983 in Wittlich, Germany. He received his Abitur diploma at the Gymnasium am Turmhof in Mechernich, Germany. Immediately after that, he began the study Knowledge Engineering at Maastricht University, The Netherlands. During his study, he worked part-time as a programmer for Albert Sickler b.v. and visited Baylor University in Waco, Texas, USA to follow the FastTrac Entrepreneurial Training Program. Maarten received his B.Sc. degree Cum Laude in 2005. After graduating, he worked as a research assistant on analyzing statistical medical data until the he started the Master program at Maastricht University. An M.Sc. degree in Artificial Intelligence followed soon in 2006, after which he worked as a research assistant on the FP6 European project MyCarEvent. From January 2007 on, Maarten was employed as a Ph.D. student (AIO) at the Department of Knowledge Engineering, Maastricht University. The research resulted in several journal and conference publications and this thesis. Besides performing scientific tasks, he was engaged in teaching, being member of the faculty council, the SIKS student advisory board and the PR country team Poland. Furthermore he participated in the organization of two Computer Olympiads. Maarten also edited the proceedings of the Computer Games Workshop 2007.

SIKS Dissertation Series

1998

1 Johan van den Akker (CWI[1]) *DEGAS - An Active, Temporal Database of Autonomous Objects*

2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

4 Dennis Breuker (UM) *Memory versus Search in Games*

5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

1999

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*

2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*

3 Don Beal (UM) *The Nature of Minimax Search*

4 Jacques Penders (UM) *The Practical Art of Moving Physical Objects*

5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*

7 David Spelt (UT) *Verification Support for Object Database Design*

8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

2000

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2 Koen Holtman (TU/e) *Prototyping of CMS Storage Management*

3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*

[1] Abbreviations: SIKS – Dutch Research School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; KUB – Katholieke Universiteit Brabant, Tilburg; KUN – Katholieke Universiteit Nijmegen; OU – Open Universiteit; RUL – Rijksuniversiteit Leiden; TUD – Technische Universiteit Delft; TU/e – Technische Universiteit Eindhoven; UL – Universiteit Leiden; UM – Universiteit Maastricht; UT – Universiteit Twente, Enschede; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; UvT – Universiteit van Tilburg; VU – Vrije Universiteit, Amsterdam.

4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*
5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*
6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*
7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*
8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*
9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*
10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*
11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

2001

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*
2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*
3 Maarten van Someren (UvA) *Learning as Problem Solving*
4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*
5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*
6 Martijn van Welie (VU) *Task-Based User Interface Design*
7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*
8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*
9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*
10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*
11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

2002

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*
2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*
3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*
4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*
5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments Inhabited by Privacy-Concerned Agents*
6 Laurens Mommers (UL) *Applied Legal Epistemology; Building a Knowledge-based Ontology of the Legal Domain*
7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*
8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TU/e) *A Reference Architecture for Adaptive Hypermedia Applications*

14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

2003

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

4 Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law – A Modelling Approach*

6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*

7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

8 Yong-Ping Ran (UM) *Repair-Based Scheduling*

9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*

10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*

13 Jeroen Donkers (UM) *Nosce Hostem – Searching with Opponent Models*

14 Stijn Hoppenbrouwers (KUN) *Freezing Language: Conceptualisation Processes across ICT-Supported Organisations*

15 Mathijs de Weerdt (TUD) *Plan Merging in Multi-Agent Systems*

16 Menzo Windhouwer (CWI) *Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouse*

17 David Jansen (UT) *Extensions of Statecharts with Probability, Time, and Stochastic Timing*

18 Levente Kocsis (UM) *Learning Search Decisions*

2004

1 Virginia Dignum (UU) *A Model for Organizational Interaction: Based on Agents, Founded in Logic*
2 Lai Xu (UvT) *Monitoring Multi-party Contracts for E-business*
3 Perry Groot (VU) *A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving*
4 Chris van Aart (UvA) *Organizational Principles for Multi-Agent Architectures*
5 Viara Popova (EUR) *Knowledge Discovery and Monotonicity*
6 Bart-Jan Hommes (TUD) *The Evaluation of Business Process Modeling Techniques*
7 Elise Boltjes (UM) *Voorbeeld$_{IG}$ Onderwijs; Voorbeeldgestuurd Onderwijs, een Opstap naar Abstract Denken, vooral voor Meisjes*
8 Joop Verbeek (UM) *Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale Politiële Gegevensuitwisseling en Digitale Expertise*
9 Martin Caminada (VU) *For the Sake of the Argument; Explorations into Argument-based Reasoning*
10 Suzanne Kabel (UvA) *Knowledge-rich Indexing of Learning-objects*
11 Michel Klein (VU) *Change Management for Distributed Ontologies*
12 The Duy Bui (UT) *Creating Emotions and Facial Expressions for Embodied Agents*
13 Wojciech Jamroga (UT) *Using Multiple Models of Reality: On Agents who Know how to Play*
14 Paul Harrenstein (UU) *Logic in Conflict. Logical Explorations in Strategic Equilibrium*
15 Arno Knobbe (UU) *Multi-Relational Data Mining*
16 Federico Divina (VU) *Hybrid Genetic Relational Search for Inductive Learning*
17 Mark Winands (UM) *Informed Search in Complex Games*
18 Vania Bessa Machado (UvA) *Supporting the Construction of Qualitative Knowledge Models*
19 Thijs Westerveld (UT) *Using generative probabilistic models for multimedia retrieval*
20 Madelon Evers (Nyenrode) *Learning from Design: facilitating multidisciplinary design teams*

2005

1 Floor Verdenius (UvA) *Methodological Aspects of Designing Induction-Based Applications*
2 Erik van der Werf (UM) *AI techniques for the game of Go*
3 Franc Grootjen (RUN) *A Pragmatic Approach to the Conceptualisation of Language*
4 Nirvana Meratnia (UT) *Towards Database Support for Moving Object data*
5 Gabriel Infante-Lopez (UvA) *Two-Level Probabilistic Grammars for Natural Language Parsing*
6 Pieter Spronck (UM) *Adaptive Game AI*
7 Flavius Frasincar (TU/e) *Hypermedia Presentation Generation for Semantic Web Information Systems*
8 Richard Vdovjak (TU/e) *A Model-driven Approach for Building Distributed Ontology-based Web Applications*
9 Jeen Broekstra (VU) *Storage, Querying and Inferencing for Semantic Web Languages*

10 Anders Bouwer (UvA) *Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments*
11 Elth Ogston (VU) *Agent Based Matchmaking and Clustering - A Decentralized Approach to Search*
12 Csaba Boer (EUR) *Distributed Simulation in Industry*
13 Fred Hamburg (UL) *Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen*
14 Borys Omelayenko (VU) *Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics*
15 Tibor Bosse (VU) *Analysis of the Dynamics of Cognitive Processes*
16 Joris Graaumans (UU) *Usability of XML Query Languages*
17 Boris Shishkov (TUD) *Software Specification Based on Re-usable Business Components*
18 Danielle Sent (UU) *Test-selection strategies for probabilistic networks*
19 Michel van Dartel (UM) *Situated Representation*
20 Cristina Coteanu (UL) *Cyber Consumer Law, State of the Art and Perspectives*
21 Wijnand Derks (UT) *Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics*

2006

1 Samuil Angelov (TU/e) *Foundations of B2B Electronic Contracting*
2 Cristina Chisalita (VU) *Contextual issues in the design and use of information technology in organizations*
3 Noor Christoph (UvA) *The role of metacognitive skills in learning to solve problems*
4 Marta Sabou (VU) *Building Web Service Ontologies*
5 Cees Pierik (UU) *Validation Techniques for Object-Oriented Proof Outlines*
6 Ziv Baida (VU) *Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling*
7 Marko Smiljanic (UT) *XML schema matching – balancing efficiency and effectiveness by means of clustering*
8 Eelco Herder (UT) *Forward, Back and Home Again - Analyzing User Behavior on the Web*
9 Mohamed Wahdan (UM) *Automatic Formulation of the Auditor's Opinion*
10 Ronny Siebes (VU) *Semantic Routing in Peer-to-Peer Systems*
11 Joeri van Ruth (UT) *Flattening Queries over Nested Data Types*
12 Bert Bongers (VU) *Interactivation - Towards an e-cology of people, our technological environment, and the arts*
13 Henk-Jan Lebbink (UU) *Dialogue and Decision Games for Information Exchanging Agents*
14 Johan Hoorn (VU) *Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change*
15 Rainer Malik (UU) *CONAN: Text Mining in the Biomedical Domain*
16 Carsten Riggelsen (UU) *Approximation Methods for Efficient Learning of Bayesian Networks*
17 Stacey Nagata (UU) *User Assistance for Multitasking with Interruptions on a Mobile Device*

18 Valentin Zhizhkun (UvA) *Graph transformation for Natural Language Processing*
19 Birna van Riemsdijk (UU) *Cognitive Agent Programming: A Semantic Approach*
20 Marina Velikova (UvT) *Monotone models for prediction in data mining*
21 Bas van Gils (RUN) *Aptness on the Web*
22 Paul de Vrieze (RUN) *Fundaments of Adaptive Personalisation*
23 Ion Juvina (UU) *Development of Cognitive Model for Navigating on the Web*
24 Laura Hollink (VU) *Semantic Annotation for Retrieval of Visual Resources*
25 Madalina Drugan (UU) *Conditional log-likelihood MDL and Evolutionary MCMC*
26 Vojkan Mihajlovic (UT) *Score Region Algebra: A Flexible Framework for Structured Information Retrieval*
27 Stefano Bocconi (CWI) *Vox Populi: generating video documentaries from semantically annotated media repositories*
28 Borkur Sigurbjornsson (UvA) *Focused Information Access using XML Element Retrieval*

2007

1 Kees Leune (UvT) *Access Control and Service-Oriented Architectures*
2 Wouter Teepe (RUG) *Reconciling Information Exchange and Confidentiality: A Formal Approach*
3 Peter Mika (VU) *Social Networks and the Semantic Web*
4 Jurriaan van Diggelen (UU) *Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach*
5 Bart Schermer (UL) *Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance*
6 Gilad Mishne (UvA) *Applied Text Analytics for Blogs*
7 Natasa Jovanovic' (UT) *To Whom It May Concern - Addressee Identification in Face-to-Face Meetings*
8 Mark Hoogendoorn (VU) *Modeling of Change in Multi-Agent Organizations*
9 David Mobach (VU) *Agent-Based Mediated Service Negotiation*
10 Huib Aldewereld (UU) *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*
11 Natalia Stash (TU/e) *Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System*
12 Marcel van Gerven (RUN) *Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty*
13 Rutger Rienks (UT) *Meetings in Smart Environments; Implications of Progressing Technology*
14 Niek Bergboer (UM) *Context-Based Image Analysis*
15 Joyca Lacroix (UM) *NIM: a Situated Computational Memory Model*
16 Davide Grossi (UU) *Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems*
17 Theodore Charitos (UU) *Reasoning with Dynamic Networks in Practice*
18 Bart Orriens (UvT) *On the development and management of adaptive business collaborations*
19 David Levy (UM) *Intimate relationships with artificial partners*

20 Slinger Jansen (UU) *Customer Configuration Updating in a Software Supply Network*
21 Karianne Vermaas (UU) *Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005*
22 Zlatko Zlatev (UT) *Goal-oriented design of value and process models from patterns*
23 Peter Barna (TU/e) *Specification of Application Logic in Web Information Systems*
24 Georgina Ramírez Camps (CWI) *Structural Features in XML Retrieval*
25 Joost Schalken (VU) *Empirical Investigations in Software Process Improvement*

2008

1 Katalin Boer-Sorbán (EUR) *Agent-Based Simulation of Financial Markets: A modular, continuous-time approach*
2 Alexei Sharpanskykh (VU) *On Computer-Aided Methods for Modeling and Analysis of Organizations*
3 Vera Hollink (UvA) *Optimizing hierarchical menus: a usage-based approach*
4 Ander de Keijzer (UT) *Management of Uncertain Data - towards unattended integration*
5 Bela Mutschler (UT) *Modeling and simulating causal dependencies on process-aware information systems from a cost perspective*
6 Arjen Hommersom (RUN) *On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective*
7 Peter van Rosmalen (OU) *Supporting the tutor in the design and support of adaptive e-learning*
8 Janneke Bolt (UU) *Bayesian Networks: Aspects of Approximate Inference*
9 Christof van Nimwegen (UU) *The paradox of the guided user: assistance can be counter-effective*
10 Wauter Bosma (UT) *Discourse oriented Summarization*
11 Vera Kartseva (VU) *Designing Controls for Network Organizations: a Value-Based Approach*
12 Jozsef Farkas (RUN) *A Semiotically Oriented Cognitive Model of Knowledge Representation*
13 Caterina Carraciolo (UvA) *Topic Driven Access to Scientific Handbooks*
14 Arthur van Bunningen (UT) *Context-Aware Querying; Better Answers with Less Effort*
15 Martijn van Otterlo (UT) *The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains*
16 Henriette van Vugt (VU) *Embodied Agents from a User's Perspective*
17 Martin Op't Land (TUD) *Applying Architecture and Ontology to the Splitting and Allying of Enterprises*
18 Guido de Croon (UM) *Adaptive Active Vision*
19 Henning Rode (UT) *From document to entity retrieval: improving precision and performance of focused text search*
20 Rex Arendsen (UvA) *Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven*
21 Krisztian Balog (UvA) *People search in the enterprise*
22 Henk Koning (UU) *Communication of IT-architecture*

23 Stefan Visscher (UU) *Bayesian network models for the management of ventilator-associated pneumonia*
24 Zharko Aleksovski (VU) *Using background knowledge in ontology matching*
25 Geert Jonker (UU) *Efficient and Equitable exchange in air traffic management plan repair using spender-signed currency*
26 Marijn Huijbregts (UT) *Segmentation, diarization and speech transcription: surprise data unraveled*
27 Hubert Vogten (OU) *Design and implementation strategies for IMS learning design*
28 Ildiko Flesh (RUN) *On the use of independence relations in Bayesian networks*
29 Dennis Reidsma (UT) *Annotations and subjective machines- Of annotators, embodied agents, users, and other humans*
30 Wouter van Atteveldt (VU) *Semantic network analysis: techniques for extracting, representing and querying media content*
31 Loes Braun (UM) *Pro-active medical information retrieval*
32 Trung B. Hui (UT) *Toward affective dialogue management using partially observable markov decision processes*
33 Frank Terpstra (UvA) *Scientific workflow design; theoretical and practical issues*
34 Jeroen de Knijf (UU) *Studies in Frequent Tree Mining*
35 Benjamin Torben-Nielsen (UvT) *Dendritic morphology: function shapes structure*

2009

1 Rasa Jurgelenaite (RUN) *Symmetric Causal Independence Models*
2 Willem Robert van Hage (VU) *Evaluating Ontology-Alignment Techniques*
3 Hans Stol (UvT) *A Framework for Evidence-based Policy Making Using IT*
4 Josephine Nabukenya (RUN) *Improving the Quality of Organisational Policy Making using Collaboration Engineering*
5 Sietse Overbeek (RUN) *Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality*
6 Muhammad Subianto (UU) *Understanding Classification*
7 Ronald Poppe (UT) *Discriminative Vision-Based Recovery and Recognition of Human Motion*
8 Volker Nannen (VU) *Evolutionary Agent-Based Policy Analysis in Dynamic Environments*
9 Benjamin Kanagwa (RUN) *Design, Discovery and Construction of Service-oriented Systems*
10 Jan Wielemaker (UvA) *Logic programming for knowledge-intensive interactive applications*
11 Alexander Boer (UvA) *Legal Theory, Sources of Law & the Semantic Web*
12 Peter Massuthe (TU/e, Humboldt-Universtät zu Berlin) *Operating Guidelines for Services*
13 Steven de Jong (UM) *Fairness in Multi-Agent Systems*
14 Maksym Korotkiy (VU) *From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)*
15 Rinke Hoekstra (UvA) *Ontology Representation - Design Patterns and Ontologies that Make Sense*

16 Fritz Reul (UvT) *New Architectures in Computer Chess*
17 Laurens van der Maaten (UvT) *Feature Extraction from Visual Data*
18 Fabian Groffen (CWI) *Armada, An Evolving Database System*
19 Valentin Robu (CWI) *Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets*
20 Bob van der Vecht (UU) *Adjustable Autonomy: Controling Influences on Decision Making*
21 Stijn Vanderlooy (UM) *Ranking and Reliable Classification*
22 Pavel Serdyukov (UT) *Search For Expertise: Going beyond direct evidence*
23 Peter Hofgesang (VU) *Modelling Web Usage in a Changing Environment*
24 Annerieke Heuvelink (VU) *Cognitive Models for Training Simulations*
25 Alex van Ballegooij (CWI) *RAM: Array Database Management through Relational Mapping*
26 Fernando Koch (UU) *An Agent-Based Model for the Development of Intelligent Mobile Services*
27 Christian Glahn (OU) *Contextual Support of social Engagement and Reflection on the Web*
28 Sander Evers (UT) *Sensor Data Management with Probabilistic Models*
29 Stanislav Pokraev (UT) *Model-Driven Semantic Integration of Service-Oriented Applications*
30 Marcin Zukowski (CWI) *Balancing vectorized query execution with bandwidth-optimized storage*
31 Sofiya Katrenko (UvA) *A Closer Look at Learning Relations from Text*
32 Rik Farenhorst and Remco de Boer (VU) *Architectural Knowledge Management: Supporting Architects and Auditors*
33 Khiet Truong (UT) *How Does Real Affect Affect Affect Recognition In Speech?*
34 Inge van de Weerd (UU) *Advancing in Software Product Management: An Incremental Method Engineering Approach*
35 Wouter Koelewijn (UL) *Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling*
36 Marco Kalz (OU) *Placement Support for Learners in Learning Networks*
37 Hendrik Drachsler (OU) *Navigation Support for Learners in Informal Learning Networks*
38 Riina Vuorikari (OU) *Tags and Self-Organisation: A Metadata Ecology for Learning Resources in a Multilingual Context*
39 Christian Stahl (TU/e, Humboldt-Universtät zu Berlin) *Service Substitution – A Behavioral Approach Based on Petri Nets*
40 Stephan Raaijmakers (UvT) *Multinomial Language Learning: Investigations into the Geometry of Language*
41 Igor Berezhnyy (UvT) *Digital Analysis of Paintings*
42 Toine Bogers (UvT) *Recommender Systems for Social Bookmarking*
43 Virginia Nunes Leal Franqueira (UT) *Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients*
44 Roberto Santana Tapia (UT) *Assessing Business-IT Alignment in Networked Organizations*

45 Jilles Vreeken (UU) *Making Pattern Mining Useful*

46 Loredana Afanasiev (UvA) *Querying XML: Benchmarks and Recursion*

2010

1 Matthijs van Leeuwen (UU) *Patterns that Matter*

2 Ingo Wassink (UT) *Work flows in Life Science*

3 Joost Geurts (CWI) *A Document Engineering Model and Processing Framework for Multimedia documents*

4 Olga Kulyk (UT) *Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments*

5 Claudia Hauff (UT) *Predicting the Effectiveness of Queries and Retrieval Systems*

6 Sander Bakkes (UvT) *Rapid Adaptation of Video Game AI*

7 Wim Fikkert (UT) *Gesture interaction at a Distance*

8 Krzysztof Siewicz (UL) *Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments*

9 Hugo Kielman (UL) *Politiële gegevensverwerking en Privacy, Naar een effectieve waarborging*

10 Rebecca Ong (UL) *Mobile Communication and Protection of Children*

11 Adriaan Ter Mors (TUD) *The world according to MARP: Multi-Agent Route Planning*

12 Susan van den Braak (UU) *Sensemaking software for crime analysis*

13 Gianluigi Folino (RUN) *High Performance Data Mining using Bio-inspired techniques*

14 Sander van Splunter (VU) *Automated Web Service Reconfiguration*

15 Lianne Bodenstaff (UT) *Managing Dependency Relations in Inter-Organizational Models*

16 Sicco Verwer (TUD) *Efficient Identification of Timed Automata, theory and practice*

17 Spyros Kotoulas (VU) *Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications*

18 Charlotte Gerritsen (VU) *Caught in the Act: Investigating Crime by Agent-Based Simulation*

19 Henriette Cramer (UvA) *People's Responses to Autonomous and Adaptive Systems*

20 Ivo Swartjes (UT) *Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative*

21 Harold van Heerde (UT) *Privacy-aware data management by means of data degradation*

22 Michiel Hildebrand (CWI) *End-user Support for Access to Heterogeneous Linked Data*

23 Bas Steunebrink (UU) *The Logical Structure of Emotions*

24 Dmytro Tykhonov (TUD) *Designing Generic and Efficient Negotiation Strategies*

25 Zulfiqar Ali Memon (VU) *Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective*

26 Ying Zhang (CWI) *XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines*

27 Marten Voulon (UL) *Automatisch contracteren*

28 Arne Koopman (UU) *Characteristic Relational Patterns*

29 Stratos Idreos (CWI) *Database Cracking: Towards Auto-tuning Database Kernels*

30 Marieke van Erp (UvT) *Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval*

31 Victor de Boer (UvA) *Ontology Enrichment from Heterogeneous Sources on the Web*

32 Marcel Hiel (UvT) *An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems*

33 Robin Aly (UT) *Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval*

34 Teduh Dirgahayu (UT) *Interaction Design in Service Compositions*

35 Dolf Trieschnigg (UT) *Proof of Concept: Concept-based Biomedical Information Retrieval*

36 Jose Janssen (OU) *Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification*

37 Niels Lohmann (TU/e) *Correctness of services and their composition*

38 Dirk Fahland (TU/e) *From Scenarios to components*

39 Ghazanfar Farooq Siddiqui (VU) *Integrative modeling of emotions in virtual agents*

40 Mark van Assem (VU) *Converting and Integrating Vocabularies for the Semantic Web*

41 Guillaume Chaslot (UM) *Monte-Carlo Tree Search*

42 Sybren de Kinderen (VU) *Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach*

43 Peter van Kranenburg (UU) *A Computational Approach to Content-Based Retrieval of Folk Song Melodies*

44 Pieter Bellekens (TU/e) *An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain*

45 Vasilios Andrikopoulo (UvT) *A theory and model for the evolution of software services*

46 Vincent Pijpers (VU) *e3alignment: Exploring Inter-Organizational Business-ICT Alignment*

47 Chen Li (UT) *Mining Process Model Variants: Challenges, Techniques, Examples*

48 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*

49 Jahn-Takeshi Saito (UM) *Solving Difficult Game Positions*

50 Bouke Huurnink (UVA) *Search in Audiovisual Broadcast Archives*

51 Alia Khairia Amin (CWI) *Understanding and supporting information seeking tasks in multiple sources*

52 Peter-Paul van Maanen (VU) *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention*

53 Edgar Meij (UVA) *Combining Concepts and Language Models for Information Access*

2011

1 Botond Cseke (RUN) *Variational Algorithms for Bayesian Inference in Latent Gaussian Models*

2 Nick Tinnemeier (UU) *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language*

3 Jan Martijn van der Werf (TU/e) *Compositional Design and Verification of Component-Based Information Systems*

4 Hado van Hasselt (UU) *Insights in Reinforcement Learning Formal analysis and empirical evaluation of temporal-difference learning algorithms*

5 Base van der Raadt (VU) *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline*

6 Yiwen Wang (TU/e) *Semantically-Enhanced Recommendations in Cultural Heritage*
7 Yujia Cao (UT) *Multimodal Information Presentation for High Load Human Computer Interaction*
8 Nieske Vergunst (UU) *BDI-based Generation of Robust Task-Oriented Dialogues*
9 Tim de Jong (OU) *Contextualised Mobile Media for Learning*
10 Bart Bogaert (UvT) *Cloud Content Contention*
11 Dhaval Vyas (UT) *Designing for Awareness: An Experience-focused HCI Perspective*
12 Carmen Bratosin (TU/e) *Grid Architecture for Distributed Process Mining*
13 Xiaoyu Mao (UvT) *Airport under Control. Multiagent Scheduling for Airport Ground Handling*
14 Milan Lovric (EUR) *Behavioral Finance and Agent-Based Artificial Markets*
15 Marijn Koolen (UvA) *The Meaning of Structure: the Value of Link Evidence for Information Retrieval*
16 Maarten Schadd (UM) *Selective Search in Games of Different Complexity*