AI techniques for the game of Go

Erik van der Werf

ISBN 90 5278 445 0
Universitaire Pers Maastricht

Printed by Datawyse b.v., Maastricht, The Netherlands.

AI techniques for the game of Go

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 donderdag 27 januari 2005 om 14:00 uur

door

Erik Cornelis Diederik van der Werf

Promotor: Prof. dr. H.J. van den Herik
Copromotor: Dr. ir. J.W.H.M. Uiterwijk

Leden van de beoordelingscommissie:
Prof. dr. A.J. van Zanten (voorzitter)
Prof. dr. A. de Bruin (Erasmus Universiteit Rotterdam)
Prof. dr. K-H. Chen (University of North Carolina at Charlotte)
Dr. J.J.M. Derks
Prof. dr. E.O. Postma

SIKS Dissertation Series No. 2005-2
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research reported in this thesis was funded by the Netherlands Organisation for Scientific Research (NWO).

Preface

In the last decade Go has been an important part of my life. As a student in Delft I became fascinated by the question why, unlike Chess, computers played this game so poorly. This fascination stimulated me to pursue computer Go as a hobby and I was fortunate to share my interests with some fellow students with whom I also founded a small Go club. In the final years of my study applied physics I joined the pattern recognition group where I performed research on non-linear feature extraction with artificial neural networks. After finishing my M.Sc. thesis I decided to pursue a Ph.D. in the fields of pattern recognition, machine learning, and artificial intelligence. When the Universiteit Maastricht offered me the opportunity to combine my research interests with my interest in Go, I did not hesitate. The research led to several conference papers, journal articles, and eventually this thesis. The research presented in this thesis has benefited from the help of many persons, whom I want to acknowledge here.

First, I would like to thank my supervisor Jaap van den Herik. His tireless efforts to provide valuable feedback, even during his holidays, greatly improved the quality of the thesis. Next, many thanks to my daily advisor Jos Uiterwijk. Without the help of both of them this thesis would have never appeared.

I would like to thank the members of the search and games group. Levente Kocsis gave me the opportunity to exchange ideas even at the most insane hours. Mark Winands provided invaluable knowledge on searching techniques, and kept me up to date with the latest ccc-gossips. I enjoyed their company on various trips to conferences, workshops, and SIKS courses, as well as in our cooperation on the program MAGOG. With Reindert-Jan Ekker I explored reinforcement learning in Go. It was a pleasure to act as his advisor. Further, I enjoyed the discussions, exchanges of ideas, and game evenings with Jeroen Donkers, Pieter Spronck, Tony Werten, and the various M.Sc. students.

I would like to thank my roommates, colleagues, and former colleagues (Natascha, Evgueni, Allard, Frank, Joop, Yong-Ping, Gerrit, Georges, Peter, Niek, Guido, Sander, Rens, Michel, Joyca, Igor, Loes, Cees-Jan, Femke, Eric, Nico, Ida, Arno, Paul, Sandro, Floris, Bart, Andreas, Stefan, Puk, Nele, and Maarten) for providing me with a pleasant working atmosphere. Moreover I thank Joke Hellemons, Marlies van der Mee, Martine Tiessen, and Hazel den Hoed for their help with administrative matters.

Aside from research and education I was also involved in university politics. I would like to thank my fraction (Janneke Harting, Louis Berkvens, Joan

Muysken, Philip Vergauwen, Hans van Kranenburg, and Wiel Kusters), the members of the commission OOI, as well as the other parties of the University Council, for the pleasant cooperation, the elucidating discussions, and the broadening of my academic scope.

Next to my research topic, Go also remained my hobby. I enjoyed playing Go in Heerlen, Maastricht, and in the Rijn-Maas liga. I thank Martin van Es, Robbert van Sluijs, Jan Oosterwijk, Jean Derks, Anton Vreedegoor, and Arnoud Michel for helping me neutralise the bad habits obtained from playing against my own program.

Over the years several people helped me relax whenever I needed a break from research. Next to those already mentioned, I would like to thank my friends from Veeto, Oele, TN, Jansbrug, Delft, and Provum. In particular I thank, the VF-promovendi Marco van Leeuwen, Jeroen Meewisse, and Jan Zuidema, 'hardcore-oelegangers' Arvind Ganga and Mark Tuil, and of course Alex Meijer, with whom I shared both my scientific and non-scientific interests in Go (good luck with your Go thesis).

More in the personal sphere, I thank Marie-Pauline for all the special moments. I hope she finds the right answers to the right questions, and, when time is ripe, I wish her well in writing her thesis. Finally, I am grateful to my parents and sister who have always supported me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 AI and games

Since the founding years of Artificial Intelligence (AI) computer games have been used as a testing ground for AI algorithms. Many game-playing systems have reached an expert level using a search-based approach. In Chess this approach achieved world-class strength, which was underlined by the defeat of World Champion Kasparov in the 1997 exhibition match against Deep Blue [158]. Go is a notable exception to this search-based development.

Go is a popular board game, played by an estimated 25 to 50 million players, in many countries around the world. It is by far the most complex popular board game, in the class of two-player perfect-information games, and has received significant attention from AI research. Yet, unlike for games such as Chess, Checkers, Draughts, and Othello, there are no Go programs that can challenge a strong human player [126].

1.2 Computer Go

The first scientific paper on computer Go was published in 1962 [144]. The first program to defeat an absolute beginner was by Zobrist in 1968, who in 1970 also wrote the first Ph.D. thesis [204] on computer Go. Since then computer Go became an increasingly popular research topic. Especially in the mid 1980s, with the appearance of cheap personal computers, and a million-dollar prize for the first computer program to defeat a professional Go player offered by Mr. Ing, research in computer Go received a big boost. Unfortunately Mr. Ing died in 1997 and despite of all efforts the prize expired at the end of 2000 without any program ever having come close to professional or even strong amateur level [126].

After Chess, Go holds a second place as test bed for game research. At recent conferences such as CG2002 [157] and ACG10 [85] the number of publications on Go was at least at a par with those on Chess. Yet, despite all efforts invested

into trying to create strong Go-playing programs, the best computer programs are still in their infancy compared to Go grandmasters. Partially this is due to the complexity [115, 147] of 19×19 Go, which renders most brute-force search techniques useless. However, even when the game is played on the smaller 9×9 board, which has a complexity between Chess and Othello [28], the current Go programs perform nearly as bad.

It is clear that the lessons from computer Chess were in its own not sufficient to create strong Go programs. In Chess the basic framework is a minimax-type searcher calling a fast and cheap evaluation function. In Go nobody has as yet come up with a fast and cheap evaluation function. Therefore, most top Go programs are using a completely opposite approach. Their evaluation functions tend to be slow and complex, and rely on many fast specialised goal-directed searches. As a consequence chess programmers are often surprised to hear that Go programs only evaluate 10 to 20 positions per second, whereas their chess programs evaluate in the order of millions of positions per second.

Although computers are continuously getting faster it is unlikely that chess-like searchers alone will suffice to build a strong Go program at least in the near future. Nevertheless, searching techniques should not be dismissed.

A direct consequence of the complexity of the evaluation functions used in computer Go is that they tend to become extremely difficult to maintain when the programmers try to increase the playing strength of their programs. The problem is that most programs are not able to acquire the Go knowledge automatically, but are instead supported by their programmers' Go skills and Go knowledge. In principle, a learning system should be able to overcome this problem.

1.3 Problem statement and research questions

From the previous sections it is clear that computer Go is a domain which has several elements that are interesting for AI research. Especially the fact that after more than thirty years of research the best computer programs still compete at only a moderate human amateur level, underlines the challenge for AI. Our main problem statement therefore is:

How can AI techniques be used to improve the strength of Go programs?

Although many AI techniques exist that might work for computer Go, it is impossible to try them all within the scope of one Ph.D. thesis. Restricting the scope we focus on two important lines of research that have proved their value in domains which we believe are related to and relevant for the domain of computer Go. These lines are: (1) searching techniques, which have been applied successfully in games such as Chess, and (2) learning techniques from pattern recognition and machine learning, which have been successful in other games, such as Backgammon, and in other complex domains such as image recognition.

This thesis will therefore focus on the following two research questions.

1. *To what extent can searching techniques be used in computer Go?*
2. *To what extent can learning techniques be used in computer Go?*

1.4 Thesis outline

The thesis is organised as follows. The first chapter is a general introduction to the topics of the thesis. Chapter 2 introduces the reader to the game of Go. The following eight chapters are split into two parts.

Chapters 3, 4, and 5 form the first part; they deal with searching techniques. Chapter 3 starts by introducing the searching techniques. In chapter 4 we investigate searching techniques for the task of solving the capture game, a simplified version of Go aimed at capturing stones, on small boards. In chapter 5 we extend the scope of our searching techniques to Go, and apply them to solve the game on small boards.

Chapters 6, 7, 8, 9, and 10 form the second part; they deal with learning techniques. Chapter 6 starts by introducing the learning techniques. In chapter 7 we present techniques for learning to predict strong professional moves from game records. Chapter 8 presents learning techniques for scoring final positions. In chapter 9 we extend these techniques to predict life and death in non-final positions. Chapter 10 investigates various learning techniques for estimating potential territory.

Finally, chapter 11 completes the thesis with conclusions and provides directions for future research.

Chapter 2

The game of Go

In this chapter we introduce the reader to the game of Go. First we provide a brief overview of the history of Go. Second, we explain the rules of the game. Third, we give a glossary explaining the Go terms used throughout the thesis. Readers who are already familiar with the game may skim through the text or even continue directly with chapter 3.

2.1 History of Go

The game of Go originates from China where it is known as Weiqi. According to legends it was invented around 2300 B.C. by an emperor to teach his son tactics, strategy, and concentration. The game was first mentioned in Chinese writings from Honan dating back to around 625 B.C. [12].

Around the 7^{th} century the game was imported to Japan where it obtained the name Igo (which later gave rise to the English name Go). In the 8^{th} century Go gained popularity at the Japanese imperial court, and around the 13^{th} century it was played by the general public in Japan. Early in the 17^{th} century, with support of the Japanese government, several Go schools were founded which greatly improved the playing level.

In the late 16^{th} century the first westerners came into contact with Go. It is interesting to note that the German philosopher and mathematician Leibniz (1646 to 1716) published an article entirely on Go even though he did not know all the rules [6, 114]. In the beginning of the 20^{th} century the inclusion of Go in a book by Edward Lasker [113], a well-known chess player (and cousin of chess world champion Emanuel Lasker), stimulated popularity in the West.

By 1978 the first western players reached the lowest professional ranks, and recently, in 2000, Michael Redmond was the first westerner to reach the highest professional rank of 9 dan. Worldwide, Go is now played by 25 to 50 million players in many countries, of which several hundreds are professional. Although most players are still located in China, Korea, and Japan, the last decades have brought a steady increase in the rest of the world, which is illustrated by the

fact that Europe nowadays has over one thousand active players of amateur dan level, and even a few professionals.

2.2 Rules

The game of Go is played by two players, Black and White, who consecutively place a stone of their colour on an empty intersection of a square grid. Usually the grid contains 19×19 intersections. However the rules are flexible enough to accommodate any other board size.

Directly neighbouring (connected) stones of the same colour form blocks. Stones remain fixed throughout the game unless their whole block is captured. In Figure 2.1 eight numbered blocks are shown as an example. (Notice that diagonal connections are not used.)

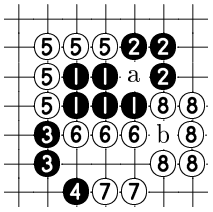


Figure 2.1: Blocks.

The directly neighbouring empty intersections of (blocks of) stones are called liberties. A block is captured and removed from the board when the opponent places a stone on its last liberty. For example, a white move in Figure 2.1 on the empty intersection marked *a* captures block 1. Moves that do not capture an opponent block and leave their own block without a liberty are called suicide. An example of suicide in Figure 2.1 would be a black move on the empty intersection marked *b*. Under most rule sets suicide is illegal, however if suicide is legal the own block is removed.

At the start of the game the board is empty. Normally the weaker player plays Black and starts the game. When the difference in strength between the players is large this can be compensated for by allowing Black to start with some additional stones (called handicap stones) placed on the empty board. For amateur players, the difference in grades of their rating (in kyu/dan) directly indicates the number of handicap stones that provides approximately equal winning chances between two players.

As the game progresses, by the alternating placement of black and white stones, the players create stable regions that are either controlled by Black or by White. A player is allowed to pass. In the end the player who controls most intersections of the board wins the game.

Although all major rule sets agree on the general idea stated above there exist several subtle differences [33]. The main differences deal with the ko rule (repetition), life and death, suicide, and the scoring method at the end of the game, which will all be discussed in the next subsections.

2.2.1 The ko rule

Since stones can be captured (removed from the board) it is possible to repeat previous board positions. However, infinite games are not practical, and therefore repetition of positions should be avoided. The most common case of a

repeating position is the basic ko, shown in Figure 2.2, where Black captures the marked white stone by playing at *a* after which White recaptures the black stone by playing a new stone at *b*. The basic-ko rule says that a move may not capture a single stone if this stone has captured a single stone in the last preceding move. As a consequence White can only recapture at b in Figure 2.2 after playing a threatening move elsewhere (which has to be answered by the opponent) to change the whole-board position. Such a move is called a ko threat.

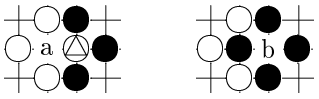


Figure 2.2: Basic ko.

Repetition in long cycles

The basic-ko rule applies under all rule sets and effectively prevents direct recreation of a previous whole-board position in a cycle of two moves (one move by Black, and one move by White). However, the basic-ko rule does not prevent repetitions in longer cycles. A simple example of repetition in a longer cycle is the triple ko (three basic kos), but more complex positions with cycles of arbitrary length exist. Although such positions are quite rare (a reasonable estimate is that they influence the result of strong 19×19 games only once every 5,000 to 50,000 games [95]) they must be dealt with if they occur in a game.

In general there are two approaches for dealing with long cycles. The first approach, which is found in traditional Asian rule sets, is to prevent only unbalanced cycles, where one side is clearly abusing the option to stay in the cycle. (This typically happens when one side refuses to pass while the other side is passing in each cycle.) For balanced cycles the traditional Asian rules have several possible rulings such as 'draw', 'no result', or adjudication by a referee. The second approach prevents long cycles by making any repetition illegal. Rules that make any repetition illegal are called super-ko rules. In general super-ko rules are found in modern western rule sets. Unfortunately, there is no agreement (yet) among the various rule sets on the exact implementation of super-ko rules, or even if they should be used at all.
In practice there are two questions that must be answered.

1. When is a position a repetition?

   First of all, for a position to be a repetition the arrangement of the stones must be identical to a previous position. However, there are more issues to consider than just the stones. We mention: the player to move, the points illegal due to the basic-ko rule, the number of consecutive passes, and the number of prisoners. When only the arrangement of stones on the board is used to prevent repetition the super-ko rule is called *positional*, otherwise it is called *situational*.

2. What are the consequences of the repetition?

   The first approach does not use super ko. Here we consider the Japanese Go rules. The Japanese Go rules [135] state that when a repetition occurs, and if both players agree, the game ends without result. In the case of

'no result' humans normally replay the game. However if time does not permit this (for instance in a tournament) the result of the game can be treated as a draw (jigo). If players play the same cycle several times but do not agree to end the game, then as an extension to the rule they are considered to end it without result [94].

For most purposes in computer Go 'no result' is not an option. Therefore, if such repeated positions occur under traditional rules, they are scored drawn unless one side captured more stones in the cycle. In that case the player that captured the most wins the game. This is identical to scoring the cycle on the difference in number of passes. The reason is that, since the configuration of stones on the board is identical after one cycle, any non-pass move must result in a prisoner, and if both sides would repeat the cycle sufficiently long one player could give away the whole board while still winning on prisoners, which count as points under the Japanese rules.

The second approach uses super ko and declares all moves illegal that recreate a previous position. The effect on the game tree is equivalent to saying that the first player to repeat a position directly loses the game with a score worse than the maximum loss of board points (any other move that does not create repetition is better).

The simplest super-ko rule is the positional super-ko rule, which only considers the arrangement of the stones on the board to determine a repetition. It has an elegant short rule text, and for that reason mathematically oriented Go enthusiasts often favour it over the traditional Asian approach. Unfortunately, however, simple super-ko rules can create strange (and by some considered unwanted) side effects. An example of such a side effect, on a small 5×5 board using positional super ko, is shown in Figure 2.3a where White can capture the single black stone by playing at the point marked *a*, which leads to Figure 2.3b. Now Black cannot recapture the ko because of the positional super-ko rule, so White is safe. If however Black would have had a slightly different position, as in Figure 2.3c, he[1] could fill one of his own eyes (which is normally considered a bad move) only to change the position and then recapture.

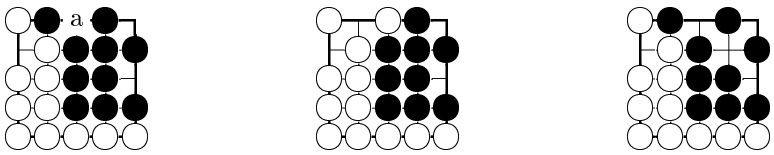


Figure 2.3: A rare side effect of positional super-ko.

[1]Throughout the thesis words such as 'he', 'she', 'his', and 'her' should be interpreted gender-neutral unless when this is obviously incorrect from the context.

In general, using a more complex situational super-ko rule can avoid most (unwanted) side effects. It should however be noted that even situational super ko is not free of unwanted, or at least unexpected, side effects (as will be shown in chapter 5).

It is interesting to note that repetition created by a pass move is never declared illegal or drawn because passing at the end of the game must be legal. As a generalisation from this kind of repetition some people have proposed that the pass move could be used to lift the ko-ban [165], i.e., repetition of a position before one player's pass then becomes legal for that player. Although this idea seems promising it is not yet clear whether it solves all problems without introducing new unexpected side effects that may be at odds with traditional rulings.

Ko rules used in this thesis

In this thesis we use the following three different compilations of the ko rules mentioned above.

1. **Basic ko** only prevents direct repetition in a cycle of length two. Longer cycles are always allowed. If we can prove a win (or loss), when analysing a position under the basic-ko rule, it means that all repetitions can be avoided by playing well. (Throughout this thesis the reader may safely assume as a default that only the basic-ko rule is relevant, unless explicitly stated otherwise.)

2. **Japanese ko** is an extension of the basic-ko rule where repetitions that are not handled by the basic-ko rule are scored by the difference in number of pass moves in one cycle. For Japanese rules this is equivalent to scoring on the difference in prisoners captured in one cycle (since the configuration of stones on the board is identical after one cycle any non-pass move in the cycle must result in a prisoner). Repetition takes into account the arrangement of stones, the position of points illegal due to the basic-ko rule, the number of consecutive passes, and the player to move. In this ko rule the Japanese rules most closely transpire, translating 'no result' to 'draw'.

   It should be noted that the Chinese rules [54] also allow repetitions to be declared drawn. However, that process involves a referee and is internally inconsistent with other rules stating that reappearance of the same board position is forbidden.

3. **Situational super ko (SSK)** declares any move that repeats a previous whole-board position illegal. A whole-board position is defined by the arrangement of stones, the position of points illegal due to the basic-ko rule, the number of consecutive passes, and the player to move.

这个禁止重复
感觉跟传统的
SSK 有点不同

2.2.2 Life and death

In human games, the life and death of groups of stones is normally decided by agreement at the end of the game. In most cases this is easy because a player only has to convince the opponent that the stones can make two eyes (see section 2.3), or that there is no way to prevent stones from being captured. If players do not agree they have to play out the position. For computers, agreement is not (yet) an option, so they always have to play out or prove life and death. In practice it is done by playing until the end where all remaining stones that cannot be proved dead statically are considered alive. (If computer programs or their operators do not reach agreement and refuse to play out the position this can cause serious difficulties in tournaments [178].)

对死活的判定依赖双方协商，但这种方法对计算机不太好使用

2.2.3 Suicide

In nearly all positions suicide is an obvious bad move. However, there exist some extremely rare positions where suicide of a block of more than one stone could be used as a ko threat or to win a capturing race. In such cases it can be argued that allowing suicide adds something interesting to the game. A drawback of allowing suicide is that the length of the game can increase drastically if players are unwilling to pass (and admit defeat). In most rule sets (Japanese, Chinese, North American, etc.) suicide is not allowed [33]. So, in all our experiments throughout the thesis suicide is illegal.

这里考虑的是不允许块子自尽

2.2.4 The scoring method

When the game ends positions have to be scored. The two main scoring methods are territory scoring and area scoring. Both methods start by removing dead stones (and adding them to the prisoners). Territory scoring, used by the Japanese rules [135], then counts the number of surrounded intersections (territory) plus the number of captured opponent stones (prisoners). Area scoring, used by the Chinese rules [54], counts the number of surrounded intersections plus the remaining stones on the board. The result of the two methods is usually the same up to one point. The result may differ when one player placed more stones than the other, for three possible reasons; (1) because Black made the first and the last move, (2) because one side passed more often during the game, and (3) because of handicap stones. Another difference between the rule sets for scoring is due to the question whether points can be counted in so-called seki positions where stones are alive without having (space for) two eyes. Japanese rules do not count points in seki. Most other rule sets, however, do count points in seki.

In all experiments throughout the thesis area scoring is used as the default and empty intersections surrounded by stones of one colour in seki are counted as points. This type of scoring is commonly referred to as Chinese scoring.

计分规则为中国规则

2.3 Glossary of Go terms

Below a brief overview of Go terms used throughout the thesis is presented. For the illustrations we generally assume the convention to hold that outside stones are always alive unless explicitly stated otherwise.

Adjacent On the Go board, two intersections are adjacent if they have a line but no intersection between them.

Alive Stones that cannot be captured are alive. Alive stones normally have two eyes or are in seki. Examples are shown in Figures 2.4 and 2.9.

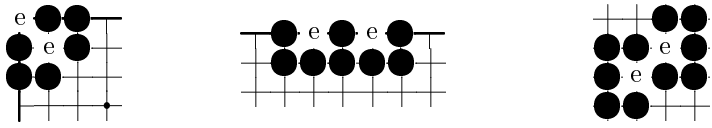


Figure 2.4: Alive stones, eyes marked *e*.

Atari Stones are said to be in atari if they can be captured on the opponent's next move, i.e., their block has only one liberty. (The marked stone in Figure 2.2 is in atari.)

Area A set of one or more intersections. For scoring, area is considered the combination of stones and territory.

Baduk Korean name for the game of Go.

Block A set of one or more connected stones of one colour. For some examples, see Figure 2.1.

Chain A set of blocks which can be connected. An example is shown in Figure 2.5.

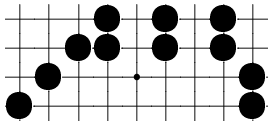


Figure 2.5: A chain of 6 blocks.

Connected Two adjacent intersections are connected if they have the same colour. Two non-adjacent intersections are connected if there is a path of adjacent intersections of their colour between them.

Dame Neutral point(s). Empty intersections that are neither controlled by Black or by White. Usually they are filled at the end of the game.

Dan Master level. For amateurs the scale runs from 1 to 6 dan, where each grade indicates an increase in strength of approximately one handicap stone. If we extrapolate the amateur scale further we get to professional level. Professional players use a more fine-grained scale where 1 dan professional is comparable to 7 dan amateur, and 9 dan professional is comparable to 9 dan amateur.

Dead Stones that cannot escape from being captured are dead. At the end of the game dead stones are removed from the board. The marked stones in Figure 2.6 are dead (they become prisoners).

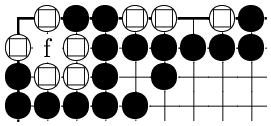


Figure 2.6: Marked stones are dead.

Eye An area surrounded by stones of one colour which provides one sure liberty. Groups that have two eyes are alive. The intersections marked *e* in Figure 2.4 are eyes.

False eye An intersection surrounded by stones of one colour which does not provide a sure liberty. False eyes connect two or more blocks which cannot connect through an alternative path. The intersection marked *f* in Figure 2.6 is a false eye for White.

Gote A move that loses initiative. Opposite of sente.

Group A (loosely) connected set of blocks of one colour which usually controls one connected area at the end of the game. The example in Figure 2.7 shows a white group that most likely controls the corner at the end of the game.

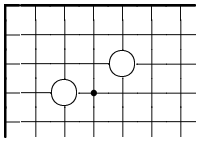


Figure 2.7: A group.

Handicap Handicap stones may be placed on the empty board by the first player at the start of the game to compensate the difference in strength with the second player. The difference in amateur grades (in kyu/dan) between two players indicates the number of handicap stones for providing approximately equal winning chances.

Jigo The result of a game where Black and White have an equal score, i.e., a drawn game.

Ko A situation of repetitive captures. See subsection 2.2.1.

Komi A pre-determined number of points added to the score of White at the end of the game. The komi is used to compensate Black's advantage of playing the first move. A commonly used value for the komi in games between players of equal strength is 6.5 . Fractional values are often used to prevent jigo.

Kyu Student level. For amateurs the scale runs from roughly 30 kyu, for beginners that just learned the rules, down to 1 kyu which is one stone below master level (1 dan). Each decrease in a kyu-grade indicates an increase in strength of approximately one handicap stone.

Liberty An empty intersection adjacent to a stone. The number of liberties of a block is a lower bound on the number of moves that has to be made to capture that block.

Ladder A simple capturing sequence which can take many moves. An example is shown in Figure 2.8.

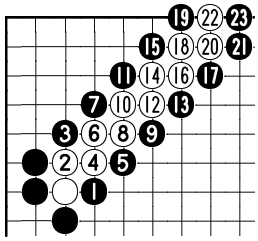


Figure 2.8: A ladder.

Life The state of being safe from capture. See also alive.

Miai Having two different (independent) options to achieve the same goal.

Prisoners Stones that are captured or dead at the end of the game.

Seki Two or more alive groups that share one or more liberties and do not have two eyes. (Neither side wants to fill the shared liberties.) Examples are shown in Figure 2.9.

Sente A move that has to be answered by the opponent, i.e., keeps the initiative. Opposite of gote.

Suicide A move that does not capture an opponent block and leaves its own block without a liberty. (Illegal under most rule sets.)

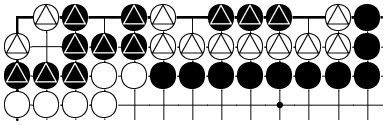


Figure 2.9: Marked stones are alive in seki.

Territory The intersections surrounded and controlled by one player at the end of the game.

Weiqi / Weichi Chinese name for the game of Go.

Chapter 3

Searching in games

This chapter gives an introduction to searching in games. First, in section 3.1 we explain the purpose of searching. Then, in section 3.2 we give an overview of the searching techniques that are commonly used in game-playing programs. Finally, in section 3.3 we discuss some fundamental questions to assess the importance of searching in computer Go.

3.1 Why search?

Go is a deterministic two-player zero-sum game with perfect information. It can be represented by a directed graph of nodes in which each node represents a possible board state. The nodes are connected by branches which represent the moves that are made between board states. From a historic perspective, we remark that in games such as Go and Chess it has become common to call this directed graph a *game tree*. It should, however, be noted that the term game tree is slightly inaccurate because it ignores the fact that some nodes may be connected by multiple paths (transpositions).

The *search tree* is that part of the game tree that is analysed by a (human or machine) player. A search tree has one root node which corresponds to the position under investigation. The legal moves in this position are represented by branches which expand the tree to nodes at a distance of one ply from the root. In an analogous way, nodes at one ply from the root can be expanded to nodes at two plies from the root, and so on. When a node is expanded d times, and positions up to d moves[1] ahead have been examined, the node is said to be investigated up to depth d. The number of branches expanding from a node is called the branching factor (the average branching factor and depth are important measures for describing the game-tree complexity). When a node is expanded the new node(s) one ply deeper are called child nodes or children.

[1] In Go the placement of a stone on the turn of one of the players is called a move. Therefore, unlike in Chess, one ply corresponds to one move.

The node one ply closer to the root is called the parent. Nodes, at the same depth, sharing the same parent are called siblings.

When nodes are not expanded, they are called leaf nodes. There are at least three reasons why leaf nodes are not expanded further: (1) the corresponding position may be final (so the result of the game is known) and the node is then often denoted as a *terminal node*, (2) there may not be enough resources to expand the leaf node any further, (3) expansion may be considered irrelevant or unnecessary.

The process of expanding nodes of a game tree to evaluate a position and find the right moves is called searching. For simple games such as Tic-Tac-Toe it is possible to expand the complete game tree so that all leaf nodes correspond to final positions where the result is known. From this it is then possible to reason backwards to construct a strategy that guarantees optimal play. In theory this can also be done for games like Go. In practice, however, the game tree is too large to expand completely because of limited computational resources. A simple estimate for the size of the game tree in Go, which assumes an average branching factor of 250 and an average game length of only 150 ply (which is quite optimistic because the longest professional games are over 400 moves), leads to a game tree of about $250^{150} \approx 10^{360}$ nodes [3] which is impossible to expand fully.

Because full expansion of game trees is impossible under nearly all circumstances, leaf nodes usually do not correspond to final positions. To let a search process deal with non-final positions evaluation functions are used to predict the value of the underlying tree (which is not expanded). In theory, a perfect evaluation function with a one-ply search (an expansion of only the root node) would be sufficient for optimal play. In practice, however, perfect evaluations are hard to construct and for most interesting games they cannot be computed within a reasonable time.

Since full expansion and perfect evaluation are both unrealistic, most search-based programs use a balanced approach where some positions are evaluated directly while others are expanded further. Balancing the complexity of the evaluation function with the size of the expanded search tree is known as the trade-off between knowledge and search [16, 84, 97, 156].

3.2 Overview of searching techniques

In the last century many techniques for searching game trees have been developed. The foundation of most game-tree search algorithms is minimax [133, 134]. Although minimax is theoretically important no modern game-playing engine uses minimax directly. Instead most game-playing engines use some form of $\alpha\beta$ search [105] which comes in many flavours. Probably the most successful flavour of $\alpha\beta$ is the iterative deepening principal variation search (PVS) [118], which is nearly identical to nega-scout [20, 31, 143]. We selected $\alpha\beta$ as the basis for all searching techniques presented in this thesis, because it is the most developed framework for searching game trees. It should however be clear that there

are many other interesting searching techniques such as B* [15, 17], BP [8], DF-PN [130], MTD(*f*) [139], OM [60, 91], PDS [129], PDS-PN [197], PN [3], PN^2 [31], PN* [162], PrOM [60], and RPS [174], which may be worth considering for building a strong Go program.

In the following subsections we will discuss the standard searching techniques that are used in this thesis. We start with minimax search (3.2.1), which provides the basis for understanding $\alpha\beta$ search discussed in subsection 3.2.2. Then in the subsequent subsections we discuss pruning (3.2.3), move ordering (3.2.4), iterative deepening (3.2.5), the transposition table (3.2.6), enhanced transposition cut-offs (3.2.7), null windows (3.2.8), and principal variation search (3.2.9).

3.2.1 Minimax search

In minimax there are two types of nodes. The first type is a max node, where the player to move (Max) tries to maximise the score. The root node (by definition ply 0) is a max node by convention, and consequently all nodes at an even ply are max nodes. The second type is a min node, where the opponent (Min) tries to minimise the score. Nodes at an odd ply are min nodes. Starting from evaluations at the leaf nodes, and by choosing the highest value of the child nodes at max nodes and the lowest value of the child nodes at min nodes, the evaluations are propagated back up the search tree, which eventually results in a value and a best move in the root node.

The strategy found by minimax is optimal in the sense that the minimax value at the root is a lower bound on the value that can be obtained at the frontier spanned by the leaf nodes of the searched tree. However, since the evaluations at leaf nodes may contain uncertainty, because they are not all final positions, this does not guarantee that the strategy is also optimal for a larger tree or for a tree of similar size after some more moves. In theory it is even possible that deeper search, resulting in a larger tree, decreases performance (unless of course when the evaluations at the leafs are not uncertain), which is known as pathology in game-tree search [131]. In practice, however, pathology does not appear to be a problem and game-playing engines generally play stronger when searching more deeply.

3.2.2 $\alpha\beta$ search

Although minimax can be used directly it is possible to determine the minimax value of a game tree much more efficiently using $\alpha\beta$ search [105]. This is achieved by using two bounds, α and β, on the score during the search. The lower bound, α, represents the worst possible score for Max. Any sub-tree of value below α is not worth investigating (this is called an α cut-off). The upper bound, β, represents the worst possible score for Min. If in a node a move is found that results in a score greater than β the node does not have to be investigated further because Min will not play this line (this is called a β cut-off).

3.2.3 Pruning

When a search process decides not to investigate some parts of the tree, which would have been investigated by a full minimax search, this is called pruning. Some pruning, such as $\alpha\beta$ pruning, can be done safely without changing the minimax result. However, it can also be interesting to prune nodes that are just unlikely to change the minimax result. When a pruning method is not guaranteed to preserve the minimax result it is called forward pruning. Forward pruning is unsafe in the sense that there is a, generally small, chance that the minimax value is not preserved. However the possible decrease in performance due the risk of missing some important lines of play can be well compensated by a greater increase in performance due to more efficient and deeper search. Two commonly used forward-pruning methods are null-move pruning [61] and multi-cut pruning [21, 198].

3.2.4 Move ordering

The efficiency of $\alpha\beta$ search heavily depends on the order in which nodes are investigated. In the worst case it is theoretically possible that the number of nodes visited by $\alpha\beta$ is identical to the full minimax tree. In the best case, when the best moves are always investigated first the number of nodes visited by $\alpha\beta$ approaches the square root of the number of nodes in the full minimax tree. (An intuitive explanation for this phenomenon is that for example to prove a win in a full game tree we have to investigate at least one move at max nodes, while investigating all moves at min nodes to check all possible refutations. In the ideal case and assuming a constant branching factor b, this then provides a branching factor of 1 at even plies and b at odd plies, which results in a tree with an average branching factor of $\sqrt{b}$ compared to b for the minimax tree.) Another way to look at this is to say that with a perfect move ordering and a limited amount of time $\alpha\beta$ can look ahead twice as deep as minimax without any extra risk.

Due to the enormous reduction in nodes that can be achieved by a good move ordering much research effort has been invested into finding good techniques for move ordering. The various move-ordering techniques can be characterised by their dependency on the search and their dependency on game-specific knowledge [106]. Well-known search-dependent move-ordering techniques are the transposition table [32], which stores the best move for previously investigated positions, the killer heuristic [2], which selects the most recent moves that generated a cut-off at the same depth, and the history heuristic [155], which orders moves based on a weighted cut-off frequency as observed in (recently) investigated parts of the search tree. In principle, these techniques are independent of game-specific knowledge. However, in Go it is possible to modify the killer heuristic and the history heuristic to improve performance by exploiting game-specific properties (as will be discussed in the next chapters).

Search-independent move-ordering techniques generally require knowledge of the game. In practice such knowledge can be derived from a domain expert.

One possible approach for this is to identify important move categories such as capture moves, defensive moves, and moves that connect. It is also possible to use learning techniques to obtain the knowledge from examples. For games like Chess and Lines of Action such an approach is described in [107] and [196], respectively. For Go this will be discussed in chapter 7.

3.2.5 Iterative deepening

The simplest implementations of $\alpha\beta$ search investigate the tree up to a predefined depth. However, it is not always easy to predict how long it takes to finish such a search. In particular when playing under tournament conditions, this constitutes a problem. Iterative deepening solves the problem by starting with a shallow search and gradually increasing the search depth (typically by one ply per iteration) until time runs out. Although this may seem inefficient at first, it actually turns out that iterative deepening can improve performance over plain fixed-depth $\alpha\beta$ [164]. The reason for this is that information from previous iterations is not lost and is re-used to improve the quality of the move ordering.

3.2.6 The transposition table

The transposition table (TT) [132] is used to store results of previously investigated nodes. It is important to store this information because nodes may be visited more than once during the search, because of the following reasons: (1) nodes may be investigated at previous iterations, (2) the same node may be reached by a different path (because the game tree is actually a directed graph), and (3) nodes may be re-searched (which will be discussed in the next subsection). The results stored in the TT typically contain information about the value, the best move, and the depth to which the node was investigated [31].

Ideally one would wish to store results of every investigated node. However, due to limitations in the available memory this is generally not possible. In most search engines the TT is implemented as a hash table [104] with a fixed number of entries. To identify and address the relevant entry in the table a position is converted to a sufficiently large number (the hash value). For this we use Zobrist hashing [205], which is the most popular hashing method among game programmers. Modern hash tables usually contain in the order of 2^{20} to 2^{26} entries, for which the lowest (20 to 26) bits of the Zobrist hash are used to address the entry. Since a search may visit many more nodes it often happens that an entry is already in use for a different position. To detect such entries a so-called lock is stored in each entry, which contains the higher bits from the Zobrist hash.

The total number of bits of the Zobrist hash (address and lock) limits the number of positions that can be uniquely identified. Since this number is always limited it is important to know the probability that a search will visit two or more different positions with the same hash, which can result in an error. A

reasonable estimate of the probability that such an error occurs is given by

$$P(errors) \approx \frac{M^2}{2N} \tag{3.1}$$

for searching M unique positions with N possible hash values, assuming M is sufficiently large and small compared to N [31]. For a search of 10^9 unique positions with a Zobrist hash of 88 bits (24 address, 64 lock) this gives a probability of about $(10^9)^2/(2 \times 2^{88}) \approx 1.6 \times 10^{-9}$ which is sufficiently close to zero. In contrast, if we would only use a hash of 64 bits the probability of getting error(s) would already be around 2.7%.

When a result is stored in an entry that is already in use for another position a choice has to be made whether the old entry is replaced. The simplest approach is always to (over)write the old one with the new result, which is called the *New* replacement scheme. However, there are several other possible replacement schemes [32] such as *Deep*, which only overwrites if the new position is searched more deeply, and *Big*, which only overwrites if the new position required searching more nodes.

In this thesis we use the *TwoDeep* replacement scheme which has two table positions per entry. If the new position is searched more deeply than the result in the first entry the first entry is moved to the second entry and the new result is stored in the first entry. Otherwise the new result is stored in the second entry. The *TwoDeep* replacement scheme always stores the latest results while preserving old entries that are searched more deeply, which are generally more important because they represent larger subtrees.

3.2.7 Enhanced transposition cut-offs

Traditionally, the TT was only used to retrieve exact results, narrow the bounds alpha and beta, and provide the first move in the move ordering. Later on it was found that additional advantage of the TT could be obtained by using enhanced transposition cut-offs (ETC) [140]. Before starting a deep search, ETC examines all successors of a node to find whether they are already stored in the TT and lead to a direct β cut-off. Especially when the first move in the normal search does not (directly) lead to a β cut-off, while another move does, investigating the latter move first can provide large savings. However, since ETC does not always provide a quick β cut-off it can create some overhead. To make up for the overhead ETC is typically used at least 2, 3, or 4 plies away from the leaves, where the amount of the tree that can be cut off is sufficiently large.

3.2.8 Null windows

In $\alpha\beta$ search the range of possible values between lower bound α, and upper bound β, is called the search window. In general, a small search window prunes more nodes than a large search window. The smallest possible search window, which contains no values between α and β, is called the null window. For a null-window search β is typically set to $\alpha + \epsilon$, with $\epsilon = 1$ for integer values. In

the case of floating point values any small value for ϵ may be used, although one has to be careful for round-off errors.

Searching with a null window usually does not provide exact results. Instead, a null-window search provides the information whether the value of a node is higher (a fail-high) or lower (a fail-low) than the bounds. In the case that a null-window search returns a value greater than α this often suffices for a direct β cut-off ($value \geq \beta$). Only in the case that an exact result is needed ($\alpha < value < \beta$) a re-search has to be done (with an adjusted search window). Although such a re-search creates some overhead the costs are usually compensated by the gains of the reduced searches that did not trigger a re-search. Moreover, the re-searches are done more efficiently because of information that is stored from the previous search(es) in the transposition table.

3.2.9 Principal variation search

In this thesis we use principal variation search (PVS) [118] as the default framework for $\alpha\beta$ search. PVS makes extensive use of null-window searches. In general it investigates all nodes with a null window unless they are on the principal variation (PV), which represents the current best line assuming best play for both sides. Although it may be possible to improve the efficiency of PVS further by doing all searches with a null window, using MTD(*f*) [139], we did not (yet) use this idea because the expected gains are relatively small compared to other possible enhancements.

The pseudo code for PVS, formulated in a negamax framework which reverses the bounds with the player to move, is shown in Figure 3.1. We note that small enhancements such as Reinefeld's depth=2 idea [143], and special code for the transposition table and move-ordering heuristics as well as various tricks to make the search slightly more efficient, are left out for clarity.

3.3 Fundamental questions

Our first research question is to what extent searching techniques can be used in computer Go. It has been pointed out by many researchers that direct application of a brute-force search to Go on the 19×19 board is infeasible. The two main reasons are the game-tree complexity and the lack of an adequate evaluation function. It therefore seems natural first to try searching techniques in a domain with reduced complexity. In Go there are two important ways to reduce the complexity of the game: (1) by decreasing the size of the board, or restricting a search to a smaller localised region; (2) by simplifying the rules while remaining to focus on tasks that are relevant for full-scale Go. For (2) there are several possible candidate tasks such as capturing, life and death, and connection games. Our investigations deal with both ways of reduction.

In the next chapter we will start with the capture game, also known as Ponnuki-Go or Atari-Go. We believe that it is an interesting test domain because it has many important characteristics of full-scale Go. This is underlined

```
PVS( alpha, beta, depth ){
  // Look up in Transposition Table
  ...
  // Narrow bounds / return value

  if( is_over() )  return( final_score() );       // Game over
  if( depth <= 0 ) return( heuristic_score() );   // Leaf node

  // Enhanced Transposition Cutoffs
  ...

  best_move = get_first_move();                   // First move
  make_move(best_move);
  best_value = -PVS( -beta, -alpha, depth-1 );
  undo_move();
  if( best_value >= beta ) goto Done;             // Beta cut-off

  move = get_next_move();                         // Other moves
  while( move != NULL ){
    alpha = max( alpha, best_value );
    make_move(move);
    value = -PVS( -alpha-1, -alpha, depth-1 );    // Null window
    if( ( alpha < value )                         // Fail high?
        && ( value < beta ) )                     // On PV?
      value = -PVS( -beta, -value, depth-1 );     // Re-search
    undo_move();
    if( value > best_value ){
      best_value = value;
      best_move = move;
      if( best_value >= beta ) goto Done;         // Beta cut-off
    }
    move = get_next_move();
  }

Done:
  // Store in Transposition Table
  ...
  // Update move ordering heuristics
  ...
  return( best_value );
}
```

Figure 3.1: Pseudo code for PVS.

by the fact that the capture game is often used to teach young children the first principles for Go. Following up on our findings for Ponnuki-Go we will then, in chapter 5, increase the complexity of the domain, by switching back to the normal rules of Go, and extend our searching techniques in an attempt to solve Go on small boards.

In both chapters our main question is:

To what extent can searching techniques provide solutions at various degrees of complexity, which is controlled by the size of the board?

Furthermore, we focus on the question:

How can we apply domain-specific knowledge to obtain an adequate evaluation function and improve the efficiency of the search?

We believe that solving the game on small boards is interesting because it is an objective way to assess the strengths and weaknesses of the various techniques. Moreover, the perfect play on small boards may later be used as a benchmark for testing other searching and learning techniques.

Chapter 4

The capture game

This chapter is based on E. C. D. van der Werf, J. W. H. M. Uiterwijk, H. J. van den Herik. Programming a computer to play and solve Ponnuki-Go. In Q. Mehdi, N. Gough, and M. Cavazza, editors, *Proceedings of GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 173–177. SCS Europe Bvba, 2002. Similar versions appeared in [185] and [186].[1]

The capture game, also known as Ponnuki-Go or Atari-Go, is a simplified version of Go that is often used to teach children the first principles of Go. The goal of the game is to be the first to capture one or more of the opponent's stones. Two rules distinguish the capture game from Go. First, capturing directly ends the game. The game is won by the side that captured the first stone(s). Second, passing is not allowed (so there is always a winner). The capture game is simpler than Go because there are no ko-fights and sacrifices,[2] and the end is well defined (capture). Nevertheless, the game still contains important elements of Go such as capturing stones, determining life and death, and making territory.

From an AI perspective solving the capture game is interesting because perfect play provides a benchmark for testing the performance of the various searching techniques and enhancements. Moreover, the knowledge of perfect play may be used to provide an absolute measure of playing strength for testing the performance of other algorithms. Since capturing stones is an essential Go skill, any algorithm that performs well on this task will of course also be of interest for computer Go.

In the following sections we present our program Ponnuki that plays the capture game using a search-based approach. The remainder of the chapter is organised as follows. Section 4.1 presents the search method, which is based

[1] The author would like to thank the editors of GAME-ON 2002, and his co-authors for the permission of reusing relevant parts of the articles in this thesis.

[2] It should be noted that the capture game is sometimes made more complex by setting the winning criterion to be the first player to capture n or more stones with $n > 1$. This allows the game to be changed more gradually towards full-scale Go because small sacrifices become possible in order to capture a big group of stones. In this chapter, however, we do not use this idea, and any capture directly ends the game.

on $\alpha\beta$ with several enhancements. Section 4.2 introduces our evaluation function. Then in section 4.3 we show the small-board solutions. It is followed by experimental results on the performance of the search enhancements and of the evaluation function. Section 4.4 presents some preliminary results on the performance of our program on larger boards. Finally, section 4.5 provides conclusions and some ideas for future work.

4.1 The search method

The standard framework for game-tree search is $\alpha\beta$ (see 3.2.2). We use PVS [118] (see Figure 3.1 and 3.2.9) with iterative deepening (see 3.2.5).

The efficiency of $\alpha\beta$ search usually improves several orders of magnitude by applying the right search enhancements. Of course, we selected a transposition table (see 3.2.6), which stores the best move, the depth, and the information about the score of previously encountered positions using the *TwoDeep* replacement scheme [32]. Moreover, we selected enhanced transposition cut-offs (ETC) [140] (see 3.2.7) to take extra advantage of the transposition table by looking at all successors of a node to find whether they contain transpositions that lead to a β cut-off before a deeper search starts. Since the ETCs are expensive we only test them three or more plies from the leaves.

4.1.1 Move ordering

The effectiveness of $\alpha\beta$ pruning heavily depends on the quality of the move ordering (see also 3.2.4). The move ordering used by PONNUKI is as follows: first the transposition move, then two killer moves [2], and finally the remainder of the moves are sorted by the history heuristic [155]. Killer moves rely on the assumption that a good move in one branch of the tree is often good in another branch at the same depth. The history heuristic uses a similar idea but is not restricted to the depth at which moves are found.

In Go it is quite common that a move on a certain intersection is good for both Black and White, which is expressed by the Go proverb "the move of my opponent is my move". This idea can be exploited in the move ordering for both the killer moves and the history heuristic. For the killer moves this is done by storing (and testing) them not only at their own depth but also one and two ply deeper. For the history heuristic we employ one table, with one entry for each intersection, which is used for ordering both the black and the white moves.

4.2 The evaluation function

The evaluation function is an essential ingredient for guiding the search towards strong play. Unlike in Chess, no good and especially no cheap evaluation functions exist for Go [28, 126]. Despite of this we tried to build an appropriate evaluation function for the capture game. The default for solving small games

is to use a three-valued evaluation function with values [∞ (win), 0 (unknown), $-\infty$ (loss)] (cf. [4]).

The three-valued evaluation function is usually quite efficient for solving small games, due to the narrow window which generates many cut-offs. However, it can become useless for strong play on larger boards when it cannot guide play towards wins that are not (yet) within the search horizon. Therefore we also developed a new heuristic evaluation function.

Our heuristic evaluation function aims at four goals:

1. maximising liberties,
2. maximising territory,
3. connecting stones, and
4. making eyes.

Naturally these four goals relate in negated form to the opponent's stones. The first goal follows directly from the goal of the game (capturing stones). Since the number of liberties is a lower bound on the number of moves that is needed to capture a stone, maximising this number is a good defensive strategy whereas minimising the opponent's liberties directly aims at winning the game. The second goal, maximising territory, is a long-term goal since it allows one side to place more stones inside its own territory (before filling it completely). The third goal follows from the observation that a small number of large groups is easier to defend than a large number of small groups. Therefore, connecting stones, which strives toward a small number of large groups, is generally a good idea. The fourth goal is directly derived from normal Go, in which eyes are the essential ingredients for building living shapes. In the capture game living shapes are only captured after one player has run out of alternative moves and is thus forced to fill his own eyes.

Since the evaluation function is used in a search tree, and thus is called in many leaves, speed is essential. Therefore our implementation uses bit-boards for fast computation of the board features. For the first two goals we use a weighted sum of the number of first-, second- and third-order liberties. (Liberties of order n are empty intersections at a Manhattan distance n from the stones). Liberties of higher order are not used since they appeared to slow down the evaluation without a significant contribution to the quality (especially on small boards). Instead of calculating individual liberties per string, the sum of liberties is directly calculated for the full board. Since the exact value for the liberties and the territory becomes quite meaningless when the difference between both sides is large the value can be clipped.

Connections and eyes are more costly to detect than the liberties. Fortunately there is a trick that combines an estimate of the two in one cheaply computable number: the Euler number [79]. The Euler number of a binary image is the number of objects minus the number of holes in those objects. Minimising the Euler number thus connects stones as well as creates eyes.

An attractive property of the Euler number is that it is a locally countable (for a proof see [79] or [120]). This is done by counting the number of occurrences of the three quad types Q_1, Q_3, and Q_d, shown in Figure 4.1, by sliding a 2×2 window over the board.

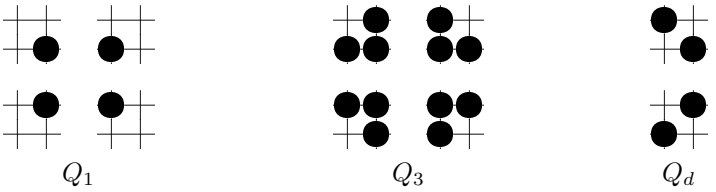
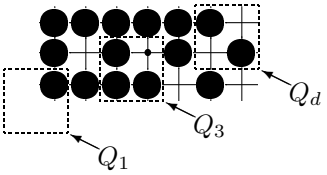


Figure 4.1: Quad types.

From the quad counts $n(Q_1)$, $n(Q_3)$, and $n(Q_d)$ we then compute

$$E = \frac{n(Q_1) - n(Q_3) + 2n(Q_d)}{4} \tag{4.1}$$

which is the zeros-joined Euler number. Zeros-joined ignores loose diagonal connections between stones (diagonal connections are used to connect the background; see below); it is the most conservative setting. More optimistic settings can be obtained by decreasing the weight of $n(Q_d)$. In the example, shown in Figure 4.2, it leads to an Euler number of 2, which corresponds to 3 objects and 1 hole. (Notice that because of the conservative zeros-joined setting only the left eye is counted as a hole. To count more eyes as holes in this position one has to decrease the weight of $n(Q_d)$ or use a non-binary approach to consider the opponent's stones in the quads too.)



$n(Q_1) = 8$, $n(Q_3) = 8$, $n(Q_d) = 4$

Figure 4.2: Quad-count example.

In our implementation we calculate two Euler numbers: one for the black stones, where we consider the white intersections as empty, and one for the white stones, where we consider the black intersections as empty. The border around the board is also taken to be empty. For speed we pre-compute quad sums per two rows, and store them in a lookup table. Consequently, during search only a small number of operations is needed.

The heuristic part of PONNUKI's evaluation function is calculated by

$$V_h = min(max(\alpha f_1 + \beta f_2 + \gamma f_3, -\delta), \delta) + \epsilon f_4 \tag{4.2}$$

in which f_1 is the number of first-order liberties for Black minus the number of first-order liberties for White, f_2 is the number of second-order liberties for Black minus the number of second-order liberties for White, f_3 is the number of third-order liberties for Black minus the number of third-order liberties for White, and f_4 is the Euler number for Black minus the Euler number for White.

The weights were set to $\alpha = 1$, $\beta = 1$, $\gamma = \frac{1}{2}$, $\delta = 3$, and $\epsilon = -4$. Won positions are evaluated by large positive values where we subtract the path length (since we prefer quick wins). For evaluating positions from the opponent's perspective we simply negate the sign.

4.3 Experimental results

This section presents results obtained on a Pentium III 1.0 GHz computer, using a transposition table with 2^{25} double entries. We discuss: (1) small-board solutions, (2) the impact of the search enhancements, and (3) the power of our evaluation function.

4.3.1 Small-board solutions

The program PONNUKI solved the empty square boards up to 5×5. Table 4.1 shows the winner, the depth (in plies) of the shortest solution, the number of nodes, and the time (in seconds) needed to find the solution, as well as the effective branching factor for each board. The principal variations for the solutions of the 4×4 and the 5×5 board are shown in the Figures 4.3 and 4.4.

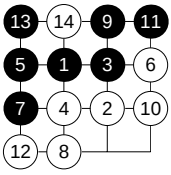


Figure 4.3: Solution for the 4×4 board.

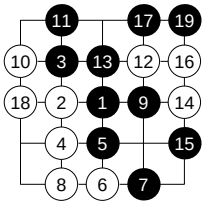


Figure 4.4: Solution for the 5×5 board.

We observed that small square boards with an even number of intersections (2×2 and 4×4) are won by the second player on zugzwang (after a sequence of moves that nearly fills the entire board the first player is forced to weaken his position because passing is not allowed). The boards with an odd number of

	2×2	3×3	4×4	5×5	6×6
Winner	W	B	W	B	?
Depth	4	7	14	19	> 23
Nodes	68	1.7×10^3	5.0×10^5	2.4×10^8	$> 10^{12}$
Time (s)	0	0	1	395	$> 10^6$
b_{eff}	2.9	2.9	2.6	2.8	

Table 4.1: Solving small empty boards.

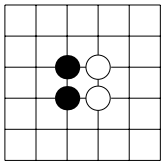


Figure 4.5: Stable starting position.

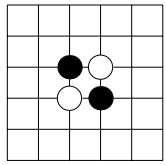


Figure 4.6: Crosscut starting position.

intersections (3×3 and 5×5) are won by the first player, who uses the initiative to take control of the centre and dominate the board. It is known that in many board games the initiative is a clear advantage when the board is sufficiently large [175]. It is therefore an interesting question whether 6×6 is won by the first or the second player. We ran our search on the empty 6×6 board for a few weeks, until a power failure crashed our machine. The results indicated that the solution is at least 24 ply deep.

Since solving the empty 6×6 board turned out a bit too difficult, we tried making the first few moves by hand. The first four moves are normally played in the centre (for the reason of controlling most territory). Normally this leads to the stable centre of Figure 4.5. An alternative starting position is the crosscut shown in Figure 4.6. The crosscut creates an unstable centre with many forcing moves. Though the position is inferior to the stable centre, when reached from the empty board, it is generally considered an interesting starting position for teaching beginners (especially on larger boards).

Around the time that we were attempting to solve the empty 6×6 board Cazenave [42] had solved the 6×6 board starting with a crosscut in the centre. His Gradual Abstract Proof Search (GAPS) algorithm, which is an interesting combination of $\alpha\beta$ with a clever threat-extension scheme, proved a win at depth 17 in around 10 minutes. Cazenave concluded that a plain $\alpha\beta$ would spend years to solve this problem. We tested PONNUKI on the same problem and found the shortest win at depth 15 in approximately 3 minutes. Figure 4.8 shows our solution for 6×6 with a crosscut. After combining our selection of search enhancements with GAPS Cazenave was able to prove the win at depth 15 in 26 seconds on an Athlon 1.7 GHz [40, 41].

	Stable	Crosscut
Winner	B	B
Depth	26 (+5)	15 (+4)
Nodes	4.0×10^{11}	1.0×10^{8}
Time (s)	8.3×10^{5}	185
b_{eff}	2.8	3.4

Table 4.2: Solutions for 6×6 with initial stones in the centre.

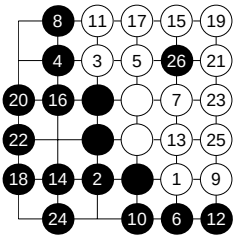


Figure 4.7: Solution for 6×6 starting with a stable centre.

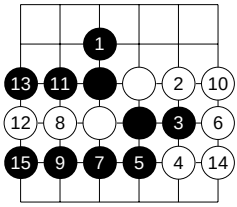


Figure 4.8: Solution for 6×6 starting with a crosscut.

Unlike the crosscut, we were not able to find quick solutions for the stable centre (Figure 4.5). (Estimates are that solving this position directly would have required around a month of computation time.) We did however prove that Black wins this position by manually playing the first move (below the two white stones). The solution is shown in Figure 4.7. The stones without numbers were placed manually, the rest was found by PONNUKI. Details of this solution are shown in Table 4.2. As far as we know Cazenave never managed to solve the stable opening, probably because there were insufficient direct threats. A number of alternative starting moves were also tested, all leading to a win for Black at the same depth, thus indicating that if the first 4 moves in the centre are correct the solution of the empty 6×6 board is a win in 31 for the first player. This supports the idea that (for boards with an even number of intersections) the initiative takes over at 6×6.

4.3.2 The impact of search enhancements

The performance of the search enhancements was measured by comparing the number of nodes searched with all enhancements to that of the search with one enhancement left out, on the task of solving the various board sizes. Results are given in Table 4.3. It is shown that on larger boards, with deeper searches, the enhancements become increasingly effective. The killer moves on the 4×4 board are an exception. The reason may be the relatively deep and narrow path leading to a win for the second player, resulting in a poor generalisation of the killers to other parts of the tree.

	3×3	4×4	5×5
Transposition tables	42%	98%	>99%
Killer moves	19%	-6%	81%
History heuristic	6%	29%	86%
Enhanced Transposition Cut-offs	0%	6%	28%

Table 4.3: Reduction of nodes by the search enhancements.

4.3.3 The power of our evaluation function

The evaluation function was compared to the standard three-valued approach for solving small trees. Usually an evaluation function with a minimal range of values generates a large number of cut-offs, and is therefore more efficient for solving small problems than the more fine-grained heuristic approaches that are needed to play on larger boards. In contrast, the results given in Table 4.4 indicate that our heuristic evaluation function outperforms the minimal approach for solving the capture game. The reason probably lies in the move ordering of which the efficiency increases with the information provided by our evaluation function.

Table 4.4 further shows that our heuristic evaluation function is quite fast. Using the heuristic evaluation function increases the average time spent per node in the search tree by only 4% compared to the three-valued approach. If we take into account that roughly 70% of all nodes were actually not directly evaluated heuristically (due to the fact that they represent illegal positions, final positions, transpositions, or are just internal nodes) this still amounts to a pure evaluation speed of roughly 5,000,000 evaluations per second. Comparing this to the over-all speed of about 600,000 nodes per second indicates that there is still significant room for adding knowledge to the evaluation function.

	heuristic		win/unknown/loss	
board	nodes	time(s)	nodes	time(s)
3×3	1.7×10^3	0	1.7×10^3	0
4×4	5.0×10^5	1	8.0×10^5	1
5×5	2.4×10^8	395	6.1×10^8	968

Table 4.4: Performance of the evaluation function.

4.4 Performance on larger boards

We tested Ponnuki (with our heuristic evaluation function) against Rainer Schütze's freeware program AtariGo 1.0 [160]. This program plays on the 10 $\times$10 board with a choice of three initial starting positions, of which one is the crosscut in the centre. Ponnuki was able to win most games, but occasionally lost when stones were trapped in a ladder. The reason for the loss was that Ponnuki used a fixed maximum depth. It did not include any means of extending ladders (which is not essential for solving the small boards). After making an ad-hoc implementation to extend simple ladders Ponnuki convincingly won all games against AtariGo 1.0.

Moreover, we tested Ponnuki against some human players too (on the empty 9×9 board). In close combat it was sometimes able to defeat reasonably strong amateur Go players, including a retired Chinese first dan. Despite of this, most stronger players were able to win easily by playing quiet territorial games. In

an informal tournament against some student programs of the study Knowledge Engineering at the Universiteit Maastricht, PONNUKI convincingly won all its games except one, which it played with drastically reduced time settings.

4.5 Chapter conclusions

We solved the capture game on the 3×3, 4×4, 5×5 and some non-empty 6×6 boards. These results were obtained by a combination of standard searching techniques, some standard enhancements that where adapted to exploit domain-specific properties of the game, and a novel evaluation function.

Regarding the first research question (see 1.3), and the questions posed in section 3.3, we may conclude that standard searching techniques and enhancements can be applied successfully for the capture game, especially when they are restricted to small regions of fewer than 30 empty intersections.

In addition, we have shown how adding inexpensive domain-specific heuristic knowledge to the evaluation function drastically improves the efficiency of the search. From the experiments we may conclude that our evaluation function performs adequately at least for the task of capturing stones.

Cazenave and our group both solved 6×6 with a crosscut using different techniques. Combining our selection of search enhancements with Cazenave's GAPS can improve the performance even further. For the capture game the next challenges are: solving the empty 6×6 board and solving the 8×8 board starting with a crosscut in the centre.

Chapter 5

Solving Go on small boards

This chapter is based on E. C. D. van der Werf, H. J. van den Herik, J. W. H. M. Uiterwijk. Solving Go on Small Boards. *ICGA Journal*, 26(2):92-107, 2003.[1]

In the previous chapter we used the capture game as a simple testing ground for the various searching techniques and enhancements. The results were quite promising owing to a novel evaluation function and domain-specific adaptations to the search enhancements. Therefore, we decided to increase the complexity of the domain by switching back to the normal rules of Go, and attempt to solve Go on small boards. Although it is difficult to scale up to large boards, and solving the 19×19 board will remain completely infeasible, we believe that searching techniques will become increasingly useful for heuristic play as well as for solving small localised regions.

Many games have been solved using a search-based approach [86]. Go is a notable exception. Up to our publication [189], the largest square board for which a computer solution had been published was the 4×4 board [161]. Although some results based on human analysis already existed for 5×5, 6×6 and 7×7 boards, they were difficult to understand and had not been confirmed by computers [55, 57, 86]. This chapter presents a search-based approach of solving Go on small boards. To support the relevance of our research we quote Davies [55].

> "*If you doubt that 5×5 Go is worthy of attention, you may be interested to know that Cho Chikun devoted over 200 diagrams to the subject in a five-month series of articles in the Japanese Go Weekly.*"

Our search method is the well-known $\alpha\beta$ framework extended with several domain-dependent and domain-independent search enhancements. A dedicated heuristic evaluation function is combined with the static recognition of unconditional territory to guide the search towards an early detection of final positions.

[1] The author would like to thank his co-authors and the editors of the *ICGA Journal* for permission to reuse relevant parts of the article in this thesis.

Our program called MIGOS (MIni GO Solver) has solved all square boards up to 5×5 and can be applied to any enclosed problem of similar size.

This chapter is organised as follows. Section 5.1 discusses the evaluation function. Section 5.2 deals with the search method and its enhancements. Section 5.3 presents an analysis of problems with super ko. Section 5.4 provides experimental results. Finally, section 5.5 gives our conclusions.

5.1 The evaluation function

The evaluation function is an essential ingredient for guiding the search towards strong play. So far there are neither good nor cheap evaluation functions for 19×19 Go [28, 126]. For small boards the situation is slightly better. In the previous chapter we introduced an adequate evaluation function for the capture game. In this chapter we use a modified version of the heuristic evaluation for the capture game (in 5.1.1), and extend it with a method for early detection of sure bounds on the final score by recognising unconditional territory (in 5.1.2). This is necessary because for Go, unlike the capture game, there is no simple criterion for deciding when the game is over (at least not until one side runs out of legal moves). Special attention is therefore given to scoring terminal positions (in 5.1.3) as well as to some subtle details about the rules (in 5.1.4).

5.1.1 Heuristic evaluation

Our heuristic evaluation function for small-board Go aims at five goals:

1. maximising the number of stones on the board,
2. maximising the number of liberties,
3. avoiding moves on the edge,
4. connecting stones, and
5. making eyes.

These goals relate in negated form to the opponent's stones. Since the evaluation function is used in tree search and is called in many leaves, speed is essential. Therefore our implementation uses bit-boards for fast computation of the board features.

Values for the first three goals are easily computed by directly counting relevant points on the board. It should be noted that instead of calculating individual liberties per block, the sum of liberties is directly calculated for the full board. For goals 4 and 5 (connections and eyes) we again use the Euler number [79], discussed in section 4.2.

Below we will not reveal all details, for reasons of competitiveness in future Go tournaments. However, we would like to state that the heuristic part of our evaluation function is implemented quite similarly to the heuristic evaluation

function discussed in section 4.2 except for two important differences. First, goal (3), which avoids moves on the edge, is new. Second, we do not use the second-order and the third-order liberties.

5.1.2 Static recognition of unconditional territory

For solving games a heuristic score is never sufficient. To be able to prove a win the highest and lowest values of the evaluation function must correspond to final positions (a sure win and a definite loss). For most games this is not a problem since the end is well defined (capture a piece, connect two sides etc.). In Go we face two problems.

The first problem is that most human games end by agreement, when both sides pass. For computers the end is usually detected by 2, 3, or 4 consecutive pass moves (the exact number of consecutive passes depends on the specific rule set). However, in nearly all positions where humans pass, a computer and especially a tree-search algorithm will try many more (useless) moves. Such moves do not affect the score and only increase the length of the game, thus pushing the final result over the horizon of any simple search.

The second problem is in the scoring itself. In even games White usually has a number of so-called komi points, which are added to White's score to compensate for the advantage of the initiative of the first player (Black). In 19×19 Go the komi is usually between 5 and 8 points. For solving the game, without knowing the komi, we have to determine the exact number of controlled points. The values for winning or losing are therefore limited by the maximum number of points on the board and should strictly dominate the heuristic scores. In Figure 5.1 we show the score range for the 5×5 board without komi. From the bottom up -1025 indicates a sure loss by 25 points, -1001 indicates a sure loss by at least one point, heuristic scores between -1000 and 1000 indicate draw or unknown, 1001 indicates a sure win by at least one point, and 1025 indicates a sure win by 25 points.

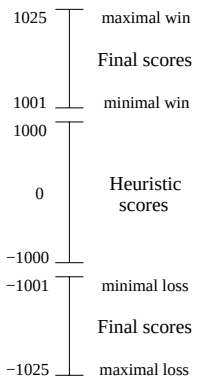


Figure 5.1: Score range for the 5×5 board.

A requirement for provably correct results when solving the game is that the winning final scores are lower bounds (the worst that can happen is that you still win with at least that score) and the losing final scores are upper bounds on the score, no matter how deep the tree is searched. For positions that are terminal (after a number of consecutive passes) this is easy, because there are no more moves. For other positions, which are closer to positions where humans would decide to pass, scoring is more difficult. To obtain reliable scores for such positions a provably correct analysis of life, death and unconditionally controlled territory is pivotal. Our analysis consists of 3 steps. First, we detect unconditional life. Second, we find the eyespace of possible unconditional territory. Third, we analyse the eyespace to see whether an invasion is possible.

Unconditional life

A set of stones is said to be unconditionally alive if they cannot be captured and never require any defensive move. A typical example of a set of unconditionally alive stones is a block with two small eyes. A straightforward approach to determine unconditional life would be to search out positions with the defending side always passing. Although such a search may have its merits it can easily become too costly.

A more elegant approach was developed by Benson [14]. His algorithm determines statically the complete set of unconditionally alive stones, in combination with a set of vital regions that form the eyespace. The algorithm is provably correct under the assumption that suicide is illegal (which is true for all major rule sets).

Benson's algorithm can be extended to determine safety under local alternating play [125]. Alternatively one could use a more refined characterisation of safety using the concept of *X life* [141]. For strong (but imperfect) play the evaluation of life and death can be further extended using heuristics such as described by Chen and Chen [45]. Although some techniques discussed here are also used by Chen and Chen [45] and by Müller [125], none of their methods are fully implemented in our program, first because heuristics do not suffice to solve the game, and second because relying on local alternating play is less safe than relying on an unconditional full-scope evaluation with global search.

Unconditional territory

Unconditional territory is defined as a set of points controlled by a set of unconditionally alive stones of the defender's colour where an invader can never build a living group, even if no defensive move is made. The set of unconditionally alive stones, as recognised by Benson's algorithm, segments the board into the following three types of regions, illustrated by 1, 2, and 3 in Figure 5.2. They form the basic elements for establishing (bounds on) the final scores.

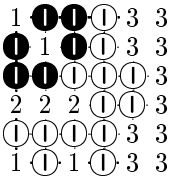


Figure 5.2: Regions to analyse.

Type 1. Benson-controlled regions are formed by the unconditionally alive blocks and their vital regions (eyespace), as classified by Benson's algorithm. The surface of these regions is unconditional and directly added to the final score.

Type 2. Open regions are not adjacent to unconditionally alive stones (which is common early in the game) or are adjacent to unconditionally alive stones of both sides. Since both sides can still be occupy the intersections of open regions they are played out further by the search.

Type 3. Closed regions are surrounded by unconditionally alive stones of one colour. They may contain stones of any colour, but can never contain

unconditionally alive stones of the invader's colour (because those would fall under type 1 or 2).

The rest of this subsection deals with classifying regions of type 3. For these regions we statically find the maximum number of sure liberties (usually eyes) an invader can make under the assumption that the defender always passes until the end of the game. If the maximum number of sure liberties is fewer than two the region is considered unconditional territory of the defending side that surrounds it (with unconditionally alive stones) and added to the final score. Otherwise it has to be played out. For region 3 in the example of Figure 5.2, which is surrounded by unconditionally alive White defender stones, it means that, since the invader (Black) can build a group with two eyes in both corners (remember the defender always passes), the territory is not unconditionally controlled by White. A one-ply search will reveal that only one defensive move of White in region 3 (as long as it is not in the corner) is sufficient to make the full region unconditional territory of White.

To determine the maximum number of sure liberties each point in the interior of the region is classified as false or true eyespace. (False eyes are completely surrounded by stones of one colour but cannot provide sure liberties because they function as a connection.) Points that are occupied by the invader's stones are not considered as possible eyespace. Determining the status of points that form possible eyespace is done by counting the number of diagonally placed unconditionally alive defender stones. If the point is on the edge and no diagonally placed unconditionally alive defender stone is present then the point can become true eyespace. If the point is not on the edge (but more towards the centre) and at most one diagonally placed unconditionally alive defender stone is present, then it can also become true eyespace. In all other cases, except one, the eyespace is false and cannot provide space for an eye. The only exception, in which a false eye is upgraded to a true eye, is when the false eye connects two regions that are already connected by an alternative path. This happens when the region forms a loop (around the unconditionally alive defender stones), which is easily detected by computing the region's Euler number.

An illustration of false eyes that are upgraded to true eyes is shown in Figure 5.3. Only Black's stones are unconditionally alive, so he is the defender, and White is the invader. All points marked f are initially false eyes of White. However, all false eyes are upgraded to true eyes, since White might play both a and b, which is possible because we assume that the defender (Black) always passes. (In practice Black will respond locally, unless there is a huge ko fight elsewhere on the board.) If Black plays a stone on a or b the loop is broken and all points marked f remain false eyes. The white stones and their neighbouring empty intersections then become unconditional territory for Black.

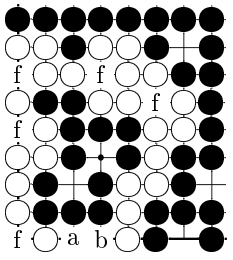


Figure 5.3: False eyes upgraded to true eyes.

Analysis of the eyespace

The analysis of the eyespace starts by looking for single defender stones. If a single defender stone is present on a false eye point then the directly neighbouring empty intersections cannot provide a sure liberty, and are therefore removed from the set of points that forms the true eyespace. (If a defender stone is connected to a second defender stone it may also remove an eye; however, this cannot be established statically and has to be determined by the search.)

Now that the true eyespace (henceforth called the eyespace) is found we test whether the eyespace is sufficiently large for two sure liberties. If the eyespace contains fewer than two points, or only two adjacent points, the territory is too small for a successful invasion and unconditionally belongs to the defender. If the eyespace is larger we continue with the analysis of defender stones inside the eyespace. Such stones may be placed to kill possible invader stones by reducing the size of the region to one single eye. In practice, we have to consider only two cases. The first case is when one defender stone is present in the invader's eyespace. The second case is when two defender stones are present in the invader's eyespace. If more than two defender stones are present in the invader's eyespace the territory can never be unconditional since the defender has to respond at least once to a capture (if he is able to prevent a successful invasion at all).

The analysis of a single defender stone is straightforward. The single defender stone contracts the stone's directly adjacent points of the invader's eyespace to a single eye, which provides one sure liberty. If the invader has no other region for eyes (non-adjacent points) any invasion fails and the territory unconditionally belongs to the defender.

The analysis of two defender stones in the invader's eyespace is harder. Here we start with noticing whether the two stones are adjacent. If the stones are non-adjacent they may provide two sure liberties; so, the territory is not unconditional. If the stones are adjacent they contract the surrounding eyespace to a two-point eye. If there is more eyespace, non-adjacent to the two defender stones, the area may provide two sure liberties for the invader and is not unconditional. If there is no more non-adjacent eyespace, the invader cannot be unconditionally alive (since he has at most one eye), but may still live in a seki together with the two defender stones.

Whether the two adjacent defender stones live in seki depends on the exact shape of the surrounding eyespace. If the two defender stones have three or fewer liberties in the eyespace of the invader, the region is too small for a seki and the area unconditionally belongs to the defender. If the two defender stones have four non-adjacent liberties and each stone directly neighbours two of the four liberties, the area can become seki. An example of such a position is shown in Figure 5.4. If the two defender stones have five liberties with more than

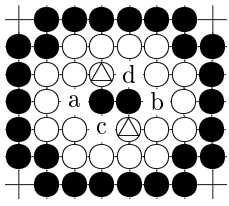


Figure 5.4: Seki with two defender stones.

one being non-adjacent, the area can become seki. If the two defender stones have six liberties the area can also become seki. Examples for five and six liberties can be obtained by removing one or two of the marked white stones in Figure 5.4. If White would play a or b the white group dies regardless of the marked stones. If one of the marked stones would move to c or d White also dies. If the area can become seki the two defender stones are counted as unconditional territory, but the rest is left undecided. If the area cannot become seki it unconditionally belongs to the defender.

5.1.3 Scoring terminal positions

In Go, games end when both sides stop placing stones on the board and play a number of consecutive passes. In all rule sets the number of consecutive passes to end the game varies between two and four. The reason why two consecutive passes do not always end the game is that the player first to pass might want to continue after the second pass. This typically occurs in positions where a basic ko is left on the board. If after two consecutive passes all moves are legal, the ko can be captured back. Therefore three or four consecutive passes are needed to end the game. The reason why under some rule sets four consecutive passes are required is that a pass can be worth a point, which is cancelled out by requiring an even number of passes. However, since passes at the end of the game do not affect area scoring, we require at most three consecutive passes.

In tree search the number of consecutive passes to end the game has to be chosen as restrictive as possible. The reason is that passes can otherwise push terminal positions over the search horizon. Thus in the case that a position contains a basic ko, and the previous position did not contain a basic ko,[2] the game ends after three consecutive passes. In all other cases two consecutive passes end the game.

Next, the terminal position has to be scored. Usually, many points on the board can be scored by recognising unconditional territory, as described in subsection 5.1.2. However, not all territory is unconditional.

For scoring points that are not in unconditional territory, dead stones must be removed from the board. This is done by counting the liberties. Each block that is not unconditionally alive and has only one liberty, which means it can be captured in one move, is removed from the board. All other stones are assumed alive. The reason why blocks with more than one liberty remain on the board is that they might live in seki, or could even become unconditionally alive after further play. If stones cannot live, or if the blocks with one liberty could have been saved, this will be revealed by (deeper) search.

Under situational super ko (SSK) (see 2.2.1) the situation is more difficult. Blocks that are not unconditionally alive and have only one liberty are sometimes not capturable because the capture would create repetition (thus making the capture illegal). Therefore, under SSK, all non-unconditional regions must be played out and all remaining stones in these regions are assumed to be alive.

[2]This additional constraint prevents repetition after two consecutive passes in rare positions such as a double ko seki (Figure 5.6).

Once the dead stones are removed, each empty point is scored based on the distance toward the nearest remaining black or white stone(s).[3] If the point is closer to a black stone it counts as one point for Black, if the point is closer to a white stone it counts as one point for White, otherwise (if the distance is equal) the point does not affect the score. The stones that remain on the board also count as points for their respective colour. Finally, the difference between black and white points, together with a possible komi, determines the outcome of the game.

5.1.4 Details about the rules

The approach used to determine statically (1) unconditional territory, and (2) (bounds on) final scores, contains four implicit assumptions about the rules. The assumptions depend on subtle details in the various rule texts which arise from tradition and are often not well formulated, obscured, or sometimes even omitted. Although these details are irrelevant under nearly all circumstances, we have to deal with them here for completeness. In this subsection we discuss the four assumptions somewhat informally. For a more formal description of our rules we refer to appendix A.

The first assumption is that suicide is illegal. That was already discussed in subsection 2.2.3.

The second assumption is that groups that have no space for two eyes or seki cannot live by an infinite source of ko threats elsewhere on the board. As a consequence moonshine life (shown in Figure 5.5) is statically classified dead if the surrounding stones are unconditionally alive. An example of an infinite source of ko threats is shown in Figure 5.6.

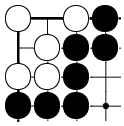


Figure 5.5: Moonshine life.

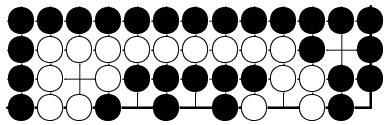


Figure 5.6: Infinite source of ko threats.

The third assumption is that groups that have possible space for two eyes or seki are not statically classified as dead. As a consequence bent four in the corner (shown in Figure 5.7) has to be played out. (Under Japanese rules the white group in Figure 5.7 is dead regardless of the rest of the board.)

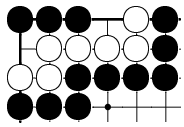


Figure 5.7: Bent four in the corner.

[3] In principle our distance-based area scoring can give slightly different results compared to other scoring methods when large open regions occur after removing dead stones of both sides based on having one liberty. However, the only position we found (so far) for which this might be considered a problem is a so-called hane seki which does not fit on the 5×5 board.

We strongly suspect that the solutions for small boards (at least up to 5 $\times$5) are independent of the second and third assumption. The reason is that an infinite source of ko threats must be separated from another group by a set of unconditionally alive stones, which just does not fit on a small board. Nevertheless the assumptions must be noted if one would apply our system to larger boards.

The fourth assumption is that capturable stones surrounded by unconditionally alive blocks are dead and the region counts as territory for the side that can capture. As a consequence in a situation such as in Figure 5.8 Black controls the whole board, even though after an actual capture of the large block White would still be able to build a living group inside the new empty region. This conflicts with one of the inconsistencies of the Japanese rules (1989) by which the white stones are considered alive (though in practice White still loses the game because of the number of captured stones).

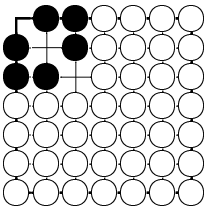


Figure 5.8: Capturable white block.

5.2 The search method

We selected the iterative-deepening principal variation search (PVS) [118] introduced in chapter 3. The efficiency of the $\alpha\beta$ search usually improves several orders of magnitude by applying the right search enhancements. We selected the following enhancements: (1) transposition tables, (2) enhanced transposition cut-offs, (3) symmetry lookups, (4) internal unconditional bounds, and (5) enhanced move ordering. All enhancements will be discussed below.

5.2.1 The transposition table

Transposition tables, introduced in subsection 3.2.6, prevent searching the same position several times by storing best move, score, and depth of previously encountered positions. Our transposition table uses the *TwoDeep* replacement scheme [32]. Iterative-deepening search with transposition tables can cause some problems with super-ko rules to be discussed in section 5.3.

5.2.2 Enhanced transposition cut-offs

We use enhanced transposition cut-offs (ETCs) [140] (see also 3.2.7) to take additional advantage of the transposition table by looking at all successors of a node to find whether they contain transpositions that lead to a direct β cut-off before a deeper search starts. Since ETCs are expensive they are only used three or more plies away from the leaves (where the amount of the tree that can be cut off is sufficiently large).

5.2.3 Symmetry lookups

The game of Go is played on a square board which contains eight symmetries. Furthermore, positions with Black to move are equal to positions where White is to move if all stones reverse colour. As a consequence these symmetries effectively reduce the state space by a factor approaching 16 (although in practice this number is significantly lower for small boards).

The effect of the use of the transposition table can be further enhanced by looking for symmetrical positions that have already been searched. In our application, when checking for symmetries, the hash keys for symmetrical positions are (re)calculated only when needed. Naturally this takes somewhat more computation time per symmetry lookup (SL), but in many positions we do not need to look up all symmetries. In fact, since most of the symmetrical positions occur more near the starting position (where also the largest node reductions can be obtained) the hashes for symmetrical positions are only computed (and used) at a distance of 5 or more plies from the leaves. When multiple symmetrical positions are found, they are all used to narrow bounds on the score.

Since all symmetrical positions have to be reached from the root it is important to check the symmetries that are likely near the root. For empty boards all symmetries are reachable; however, for non-empty boards many symmetries can be broken. Time is saved by not checking unlikely symmetries in the tree.

It should further be noted that under SSK symmetrical transpositions can only be used for move ordering (because the history is different).

5.2.4 Internal unconditional bounds

Recognising unconditional territory is important for scoring leaves. However, in many cases unconditional territory can also be used in internal nodes to improve the efficiency of the search.

The analysis of unconditional territory, presented in subsection 5.1.2, divides the board into regions that are either unconditionally controlled by one colour or are left undecided. In internal nodes, we use the size of these regions to compute two unconditional bounds on the score, which we call internal unconditional bounds (IUB). An upper bound is calculated by assigning all undecided points to friendly territory. A lower bound is calculated by assigning all undecided points to the opponent's territory. If the upper bound on the score is equal to or smaller than α, or the lower bound on the score is equal to or larger than β, the search directly generates a cut-off. In other cases the bounds can still be used to narrow the $\alpha\beta$ window, thus generating more cut-offs deeper in the tree.

Unconditional territory can further be used for reducing the branching factor. The reason is that moves inside unconditional territory normally do not have to be examined since they cannot change the outcome of the game. Exceptions are rare positions where changing the state just for the sake of changing the history for a future position is essential. Therefore all legal moves are examined under SSK.

5.2.5 Enhanced move ordering

The move ordering is enhanced by the following three heuristics: (1) history heuristic [155, 199], (2) killer moves [2], and (3) sibling promotion [66]. As in the previous chapter, all move-ordering enhancements are implemented (and modified) to utilise the Go proverb "the move of my opponent is my move".

History heuristic

The standard implementation of the history heuristic (HH) orders moves based on a weighted cut-off frequency as observed in (recently) investigated parts of the search tree. In games such as Chess it is normal to have separate tables for the moves of each player. In contrast, our Go-specific implementation of the history heuristic employs one table, sharing intersections for both the black and the white moves.

Killer moves

The standard killer moves (KM) are the moves that most recently generated a cut-off at the same depth in the search tree. Our implementation of the killer moves stores (and tests) them not only at their own depth but also one and two ply deeper. (We remark that testing KM two ply deeper in the search tree is not a new idea. However, testing them one ply deeper, where the opponent is to move, is not done in other games such as Chess.)

Sibling promotion

When a search returns (without generating a cut-off), the intersection of the opponent's expected best reply is often an interesting intersection to try next. Taking the opponent's expected reply as the next move to be investigated is called sibling promotion (SP) [66].

Since the quality of the moves proposed by the search heavily depends on the search depth, SP does not work well in nodes where the remaining search depth is shallow. Therefore, our implementation of SP is only active in nodes that are at least 5 plies away from the leaves. After our implementation of SP is called it remains active until it generates a move that is already examined or illegal, after which the move ordering proceeds to the next ordering heuristic.

Complete move ordering

The complete move ordering is as follows:

(1) the transposition move,	(2) sibling promotion,
(3) the first move sorted by the history heuristic,	(4) sibling promotion,
(5) the first killer move,	(6) sibling promotion,
(7) the second move sorted by the history heuristic,	(8) sibling promotion,
(9) the second killer move,	(10) sibling promotion,
(11) the next move sorted by the history heuristic,	(12) sibling promotion,
(13) back to (11).	

5.3 Problems with super ko

Though the use of transposition tables is pivotal for efficient iterative-deepening search it can create so-called graph history interaction (GHI) problems [36] if the SSK rule applies. The reason is that the history of a position is normally not included in the transposition table, which means that in some special cases the transposition may suggest a continuation that is illegal or sub optimal under super ko. In our experiments we found two variations of the problem which we call the *shifting-depth variant* and the *fixed-depth variant*. Below both variants are presented, in combination with some possible solutions.[4]

5.3.1 The shifting-depth variant

The shifting-depth variant of the GHI problem is illustrated by the following example. Consider an iterative-deepening search until depth n examining the following sequence of positions:

(root)-A-B-C-D-E-F-...-(heuristic score)

Now the heuristic score and the depth are stored in the transposition table for position D. Assume in the next iteration, the iterative-deepening process searches until depth $n + 1$, and examines the following sequence of positions:

(root)-J-K-L-F-...-D

Here position D is found closer to the leaves. From the information stored in the transposition table, D is assumed to be examined sufficiently deep (in the previous iteration). As a consequence the search directly returns the heuristic score from the transposition table. However, the result for D is not valid because the assumed continuation contains a repetition (F). This will not be observed since the moves are not actually made. The problem is even more disturbing because the transposition is found at another depth and in a following iteration, which means that results of the subsequent iterations can inherit a heuristic score. It is remarked that a final score would have been calculated if the moves were actually made.

To complicate matters even more, it is possible to construct positions where alternating lines of play continuously inherit heuristic scores from previous iterations through the transposition table. An example of such a position, found for the 3×3 board, is shown in Figure 5.9. Here White's move 4 is a mistake under situational super ko because Black could take control of the full board by playing 5 directly adjacent to 4. (Black 1 is of course also sub-optimal, but that is not important here.) However, the iterative-deepening search only returns a heuristic score for that line because of depth-shifted transpositions (which implicitly rely on continuations that are illegal under SSK). Black does not see the

[4] Recently a solution to the GHI problem in Go was proposed for depth-first proof-number search (DF-PN) [102], and for $\alpha\beta$ search [101]. It can be used efficiently when the complete proof-tree is stored in memory.

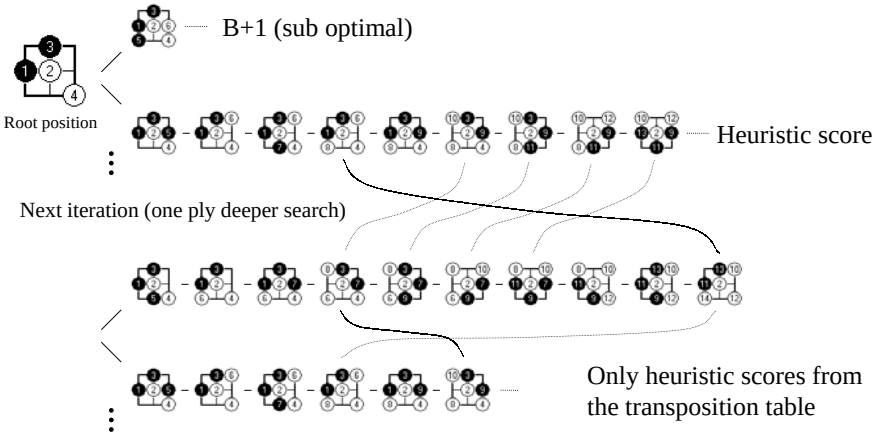


Figure 5.9: Sub-optimal under SSK due to the shifting-depth variant.

optimal win either, because of the same problem, and will also play sub-optimal obtaining only a narrow victory of one point.

To overcome shifts in depth we can incorporate the number of passes and captured stones in the full hash used to characterise positions. Then only transpositions are possible to positions found at the same depth. (Of course it is also possible to store the path length directly in the transposition table at the cost of a small amount of additional memory and speed.)

5.3.2 The fixed-depth variant

Although fixing the depth (by including passes and captures in the hash) solves most problems, it still leaves some room for errors. The fixed-depth variant of the GHI problem is illustrated by the following example. Consider an iterative-deepening search examining the following sequence of positions:

(root)-A-B-...-C-...-B

Since B is illegal because of super ko this will become:

(root)-A-B-...-C-...-D-...

Now C is stored in the transposition table. Assume after some time the search investigates:

(root)-E-F-...-C

C is found in the transposition table and was previously examined to the same depth. As a consequence this line will not be expanded further. However, the

value of C is based on the continuation C-...-D where C-...-B-... may give a higher score.

Another example of the fixed-depth GHI problem when using SSK is illustrated by Figure 5.10. Here the optimal sequence is as follows: (1-4) as shown, (5) Black captures the white stone marked 2 by playing at the marked square, (6) White passes, (7) Black at 1, (8) White passes (playing at 2 is illegal by SSK), (9) Black captures White's upper right group and wins by 17 points.

Now assume the following alternative sequence: (1) Black at 3, (2) White at 4, (3) Black at 1, (4) White at 2. The configuration of stones on the board is now identical to Figure 5.10. However, under SSK Black cannot play the same follow-up sequence and loses by 3 points.

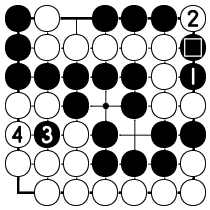


Figure 5.10: Black win by SSK.

Although both sequences of moves lead to the same position only the first sequence kills a white group. Since both sequences of moves lead to the same position at the same depth one of them, which one depends on the move ordering, can be valued incorrectly if the history is not taken into account.

In practice using separate hash keys for the number of passes and the number of captured stones, combined with a decent move ordering, is sufficient to overcome nearly all problems with super ko. However, for a proof this is not sufficient. Though it is possible to include the full history of a position in the hash, our experiments indicate a drastic decrease in search efficiency, thus making it impractical for larger problems.

The reader should note that although (depth-shifted) transpositions are a problem for SSK they are not a problem under the Japanese-ko rule. The reason is that draws never dominate a win or a loss, because draws are in the range of the heuristic scores. In practice we use 0 as the value for a draw, and we do not distinguish between a draw and heuristic scores. (Alternatively one could use 1000 (or -1000) to indicate that Black (or White) could at least achieve a draw, which might be missed because of depth-shifted transpositions.) Occasionally depth-shifted transpositions are even useful, for solving positions under basic or Japanese ko, because they enable the search to look beyond the fixed depth of the iteration.

5.4 Experimental results

This section presents the results obtained by a Pentium IV 2.0 GHz computer, using a transposition table with 2^{24} double entries (for the *TwoDeep* replacement scheme) and a full Zobrist hash of 88 bits. We discuss: (1) small-board solutions, (2) opening moves on the 5×5 board, (3) the impact of recognising unconditional territory, (4) the power of the search enhancements, (5) preliminary results for the 6×6 board, and (6) scaling up.

board	ko rule	Move	Result	Depth	Nodes (log_{10})	Time	b_{eff}
2×2	basic	a1	0	5	2.1	n.a.	2.65
	Japanese	a1	0	5	2.1	n.a.	2.65
	appr. SSK	a1	+1	11	2.9	n.a.	1.83
	full SSK	a1	+1	11	3.1	n.a.	1.91
3×3	basic	b2	+9	11	3.5	n.a.	2.06
	Japanese	b2	+9	11	3.5	n.a.	2.06
	appr. SSK	b2	+9	11	4.0	n.a.	2.30
	full SSK	b2	+9	11	4.4	n.a.	2.51
4×4	basic	b2	+1	21	5.8	3.3 (s)	1.90
	Japanese	b2	+2	21	5.8	3.3 (s)	1.90
	appr. SSK	b2	+2	23	6.9	14.8 (s)	1.99
	full SSK	b2	+2	23	9.5	1.1 (h)	2.58
5×5	basic	c3	+25	23	9.2	2.7 (h)	2.51
	Japanese	c3	+25	23	9.2	2.7 (h)	2.51
	appr. SSK	c3	+25	23	10.0	9.7 (h)	2.73

Table 5.1: Solving small empty boards.

5.4.1 Small-board solutions

MIGOS solved the empty square boards sized up to 5×5.[5] Table 5.1 shows the ko rule, the best move, the result, the depth (in plies) where the PV becomes stable, the number of nodes, the time needed to find the solution, and the effective branching factor for each board. (In column 'time', s means seconds and h hours.)

The reader should note that 'depth' here does not mean the maximum length of the game (the losing side often can make some more futile moves). It just means that after that depth the Principal Variation and value of the move were no longer affected by deeper searches. As a consequence, boards that did not get a maximal score (e.g., 2×2 and 4×4) could in principle contain an undetected deep variant that might raise the score further. To rule out the possibility of a higher score both boards were re-searched with adjusted komi. The komi was adjusted so that it converted the loss of the second player to a win by one point. Finding the win then established both the lower and the upper bound on the score, thus confirming that they are indeed correct. Our results for boards up to 4×4 confirm the results published in [86], which apparently assumed Chinese scoring with a super-ko rule. Since the two-point victory for Black on the 4×4 board is a seki (which is a draw under Japanese rules) it also confirms the results of [161]. For all square boards up to 5×5, our results mutually confirm results based on human analysis [62, 172].

Table 5.1 shows results for two possible implementations of SSK. Approximate SSK does not check the full history, but does prevent most common problems by including the number of passes and the number of captured stones

[5] We would like to remind the reader that winning scores under basic ko are lower bounds for the score under SSK (because repetition can be avoided by playing well). Therefore the empty 5×5 board is solved regardless of super ko.

in the hash. Full SSK stores (and checks) a separate hash for the history in all entries of the transposition table. Both implementations of SSK require significantly more nodes than Japanese ko. In particular, full SSK requires too much effort to be practical for larger boards.

It is interesting that under SSK Black can win the 2×2 board by one point. The search for this tiny board requires 11 plies, just as deep as solutions for the 3×3 board! Another conspicuous result is that under basic ko Black does not win the 4×4 board by two points. The reason is that the two-point victory is unreachable by a seki with a cycle where White throws in more stones than Black per cycle.

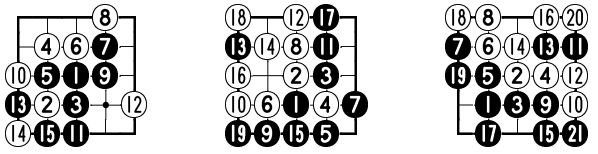
5.4.2 Opening moves on the 5×5 board

We analysed all opening moves for the 5×5 board. An opening in the centre leads to an easy win for Black, as shown in Figure 5.11a. However, alternative openings are much more challenging to solve. In Figure 5.12 the results are shown, with the numbered stones representing the winner (by the colour) and the score (by the number), for all possible opening moves.

Most difficult of all is the opening move on the b2 point. It is lost by one point only. After White's response in the centre MIGOS still required a 36-ply deep search (which took some days) to find the one-point victory. Proving that White cannot do better takes even longer. Optimal play for the b2 opening is shown in Figure 5.11c. Although this line of play is 'only' 21 ply deep, both Black and White can easily force much deeper variations. A typical example is when White throws in 16 at 19, followed by Black at 16, which leads to a draw. If Black plays 11 at 15 he loses the full board at ply 40.

The opening on c2 is also challenging, requiring a 28-ply deep search. Optimal play for the c2 opening is shown in Figure 5.11b. Extending with White 6 at 7 is a mistake and leads to a seki won with 5 points by Black (a mistake that was overlooked even by Cho Chikun [56]).

The results for both alternative opening moves support the main lines of the human solutions by Ted Drange, Bill Taylor, John Tromp, and Bill Spight [172]. (However, we did find some subtle differences deep in the tree, due to differences in the rules.)



(a) (b) (c)

Figure 5.11: Optimal play for central openings.

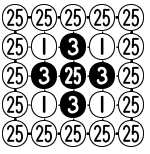


Figure 5.12: Values of opening moves on the 5×5 board.

5.4.3 The impact of recognising unconditional territory

In subsection 5.1.2 we introduced a method for static recognition of unconditional territory (UT). It is used to detect final positions, or generate cut-offs, as soon as possible (thus avoiding deeper search until both players eventually must pass or create repetition). Although our method reduces the search depth it does not necessarily mean that the search is always more efficient. The reason is that static analysis of unconditional territory is more expensive than a simple evaluation at the leaves (recognising dead stones by having only one liberty, or playing it out completely). To test the impact on the performance of recognising unconditional territory we used our program to solve the small empty boards without recognising unconditional territory, and compared it to the version that did recognise unconditional territory.

board	Use UT	Depth	Nodes (log_{10})	Time	Speed (knps)	b_{eff}
3×3	+	11	3.5	n.a.	n.a.	2.06
3×3	-	13	3.8	n.a.	n.a.	1.97
4×4	+	21	5.8	3.3 (s)	214	1.90
4×4	-	25	6.4	12.7 (s)	213	1.80
5×5	+	23	9.2	2.7 (h)	160	2.51
5×5	-	>30	>10.7	>2 (d)	$\sim$ 340	

Table 5.2: The impact of recognising unconditional territory.

The results, shown in Table 5.2, indicate that recognising unconditional territory reduces the search depth, the time, and the number of nodes for solving the small boards. (In column 'time', s means seconds, h hours and d days.) Although the speed in nodes per second is significantly less on the 5×5 board the reduced depth easily compensates this.

On the 4×4 board we observed no significant speed difference in nodes per second. For this there are at least four reasons: (1) most 4×4 positions cannot contain unconditionally alive blocks (therefore a large amount of costly analysis is often not performed), (2) many expensive evaluations are retrieved from a cache, (3) due to initialisation time the results for small searches may be inaccurate, and (4) the search trees are different (different positions require different time).

5.4.4 The power of search enhancements

The performance of the search enhancements was measured by comparing the reduction in the number of nodes between a search using all enhancements and a search with one enhancement left out. The results, on the task of solving the various board sizes, are given in Table 5.3.

It is shown that on larger boards, with deeper searches, the enhancements become increasingly effective. The transposition table is clearly pivotal. Enhanced transposition cut-offs are quite effective, although the results suggest a

	3×3	4×4	5×5
Transposition tables	92 %	>99 %	>99 %
Enhanced transposition cut-offs	4 %	35 %	40 %
Symmetry lookups	63 %	89 %	86 %
Internal unconditional bounds	5 %	1 %	7 %
History heuristic	61 %	90 %	95 %
Killer moves	0 %	8 %	20 %
Sibling promotion	0 %	3 %	31 %

Table 5.3: Reduction of nodes by the search enhancements.

slightly overestimated importance (because we neglect time). Symmetry lookups are quite useful, at least on the empty boards. Internal unconditional bounds are not really effective because unconditional territory is also recognised at the leaves (on larger boards it may be interesting to turn off unconditional territory at the leaves and only use internal unconditional bounds). Our implementation of the history heuristic (one table for both sides) is very effective compared to the killer moves. This is also the reason why the first history move is examined before the killer moves. Finally, sibling promotion works quite well, especially on larger boards.

5.4.5 Preliminary results for the 6×6 board

After solving the 5×5 board we tried to solve the 6×6 board. MIGOS did not solve the empty 6×6 board. However, based on human solutions it is possible to make the first few moves by hand. The most difficult position for which MIGOS proved a win, is shown in Figure 5.13. After about 13 days, searching 220 billion nodes in 25 ply, it proved that Black wins this position by at least two points. However, we were not able to prove the exact value (which is expected to be a win by 4 points).

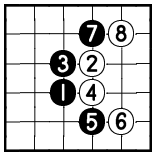


Figure 5.13: Black win (≥2).

We tested MIGOS on a set of 24 problems for the 6×6 board published in *Go World* by James Davies [52, 53]. For 21 problems it found the correct move, for the other 3 it found a move that is equally good (at least for Chinese scoring). The correct moves usually turned up within a few seconds; solving the positions (i.e., returning a final score) took more time. MIGOS solved 19 problems of which two only reached at least a draw (probably because of the occasional one-point difference between Japanese and Chinese scoring). All problems with more than 13 stones were solved in a few minutes or seconds only. The problems that were not solved in a couple of hours had 10 or less stones on the board.

5.4.6 Scaling up

As computers are becoming more powerful over time, searching techniques tend to become increasingly powerful as well. Primarily this is, of course, caused by the increasing processor clock speed(s) which directly improve the raw speed in nodes per second. Another important factor is the increase in available working memory, which affects the speed in nodes per second (through various caches) as well as the number of nodes that have to be investigated (through the transposition table). To obtain an indication of how both factors reduce time, we re-tuned and tested our search on a number of old machines. The results, shown in Table 5.4 for solving the 4×4 board and in Table 5.5 for solving the 5×5 board, indicate that the amount of memory is not so important on the 4×4 board. However, on the 5×5 board the increased memory gave a factor of 4 in reduction of nodes compared to the 6 year old Pentium 133MHz. We therefore expect even bigger pay-offs from increased memory for larger boards.

Machine	TT size (log_2)	Depth	Nodes (log_{10})	Time	Speed (knps)	b_{eff}
486 DX2 66MHz	18	21	5.8	182.8 (s)	3.8	1.90
Pentium 133MHz	20	21	5.8	47.8 (s)	14.6	1.90
Pentium III 450MHz	22	21	5.8	11.2 (s)	62.2	1.90
Pentium IV 2GHz	24	21	5.8	3.3 (s)	214	1.90

Table 5.4: Solving the 4×4 board on old hardware.

Machine	TT size (log_2)	Depth	Nodes (log_{10})	Time	Speed (knps)	b_{eff}
486 DX2 66MHz	18	23	10.1	55 (d)	2.5	2.74
Pentium 133MHz	20	23	9.8	6.6 (d)	11.2	2.67
Pentium III 450MHz	22	23	9.5	17.1 (h)	54.3	2.59
Pentium IV 2GHz	24	23	9.2	2.7 (h)	160	2.51

Table 5.5: Solving the 5×5 board on old hardware.

5.5 Chapter conclusions

The main result is that MIGOS solved Go on the 5×5 board for all possible opening moves. Further, the program solved several 6×6 positions with 8 and more stones on the board. The results were reached by a combination of standard (TT, ETC), improved (HH, KM, SP), and new (IUB, SL) search enhancements, a dedicated heuristic evaluation function, and a method for static recognition of unconditional territory.

So far only the 4×4 board was solved by Sei and Kawashima [161]. For this

board their search required 14,000,000 nodes. MIGOS was able to confirm their solutions and solved the same board in fewer than 700,000 nodes. Hence we conclude that the static recognition of unconditional territory, the symmetry lookups, the enhanced move ordering, and our Go-specific improvements to the various search enhancements are key ingredients for solving Go on small boards.

We analysed the application of the situational-super-ko rule in tree search, and compared it to the Japanese rules for dealing with repetition. For solving positions, SSK quickly becomes impractical. It is possible to obtain good approximate results by reducing the information stored about the history of a position to the number of passes and captures. However, for most practical purposes super ko is irrelevant and can be ignored safely because winning scores under basic ko are lower bounds on the score under SSK.

Regarding the first research question (see 1.3), and to answer the main question posed in section 3.3; our experience with MIGOS leads us to conclude that, on current hardware, provably correct solutions can be obtained within a reasonable time frame for confined regions of size up to about 28 intersections.

Moreover, for efficiency of the search, provably correct domain-specific knowledge is essential to obtain tight bounds on the score early in the search tree. We showed that without such domain-specific knowledge, detecting final positions by search alone becomes unreasonably expensive.

Future expectations

The next challenges in small-board Go are: solving the 6×6 and 7×7 boards. Both boards are claimed to have been solved by humans, but so far no computer was able to confirm the results. The human solutions for the 6×6 board suggests a 4 point victory for Black [172]. The 7×7 board is claimed to have been solved by a group of Japanese amateurs including Kiga Yasuo, Nebashi Teruichi, Noro Natsuo and Yamashita Isao. In 1989, after several years of work, with some professional help from Kudo Norio and Nakayama Noriyuki, they reached the conclusion that Black wins by 9 points [57].

On today's standard PC MIGOS is not yet ready to take on the empty 6×6 board. However, over the last decade we have seen an over-all speedup by almost a factor 500 for solving 5×5. A continuing increase in hardware performance alone may enable our searching techniques to solve the 6×6 boards on an ordinary PC in less than 20 years. Although we might just sit back and wait a few years, there are of course ways to speed up the process, e.g., by using a massive parallel system, or by using the human solutions to guide and extend the search selectively, or work backward from the end.

On the AI side, we believe that large gains can be expected from adding more (provably correct) Go knowledge to the evaluation function (for obtaining final scores earlier in the tree). Further, a scheme for selective search extensions, examining a highly asymmetric tree (resembling the human solutions), may enable the search to solve the 6×6 and the 7×7 boards much more efficiently than our current fixed-depth iterative-deepening search without extensions. Next to

these suggestions an improved move ordering may increase the search efficiency, possibly even by several orders of magnitude.

Acknowledgements

We are grateful to Craig R. Hutchinson and John Tromp for information on the human solutions, and to Andries Brouwer, Ken Chen, Zhixing Chen, Robert Jasiek, the anonymous referees of the *ICGA Journal*, and several others on the computer-Go mailing list and rec.games.go for pointing our attention to some subtle details in the rules and providing feedback on earlier drafts of this chapter.

Chapter 6

Learning in games

This chapter is partially based on[1]

1. E. C. D. van der Werf, J. W. H. M. Uiterwijk, and H. J. van den Herik. Learning connectedness in binary images. In B. Kröse, M. de Rijke, G. Schreiber, M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'01)*, pages 459–466. 2001.

2. E. C. D. van der Werf and H. J. van den Herik. Visual learning in Go. In J.W.H.M. Uiterwijk, editor, *The CMG Sixth Computer Olympiad: Computer-Games Workshop Proceedings Maastricht*. Technical Report CS 01-04, IKAT, Department of Computer Science, Universiteit Maastricht, 2001.

This chapter provides an introduction to learning in games. First, in section 6.1 we explain the purpose of learning. Then, in section 6.2 we give an overview of the learning techniques that can be used for game-playing programs. In section 6.3 we discuss some fundamental questions to assess the importance of learning in computer Go. Finally, in section 6.4 we explore techniques for the fundamental task of learning connectedness.

6.1 Why learn?

In the previous chapter we have shown how searching techniques can be used to play well on small boards. As the size of the board grows knowledge becomes increasingly important, which was underlined by the experiments assessing the importance of recognising unconditional territory. The evaluation function of Migos uses both heuristic and provably correct knowledge to play well on small boards. Due to limitations in our understanding of the game as well as limitations inherently imposed by the complexity of the game, improving the program

[1]The author would like to thank his co-authors for the permission of reusing relevant parts of the articles in this thesis.

with additional hand-coded Go knowledge tends to become increasingly difficult. In principle, a learning system should be able to overcome this problem, at least for adding heuristic knowledge.

The main reason why we are interested in learning techniques is to lift the programmers' burden of having to acquire, understand, and implement all Go knowledge manually. However there are more reasons: (1) learning techniques have been successful in other games such as Backgammon [170], (2) learning techniques have, at least to some extent, been successful in related complex domains such as image recognition, and (3) understanding of learning techniques that are successful for computer Go may provide some insights into understanding human intelligence and in particular why humans are able to play Go so well. This said, we would like to stress that our learning techniques are by no means a model of how human learning is assumed to take place. Although we, and many other researchers in the fields of neural networks, pattern recognition, and machine learning, may have been inspired by models of the human brain and visual system, our learning techniques are best understood from the mathematical perspective of general function approximators. The primary goal of our research is to obtain a good performance at tasks that are relevant for playing Go. In this context, issues such as biological plausibility are only relevant to the extent that they may provide some hints on how to obtain a good performance.

6.2 Overview of learning techniques

In the last century many techniques for learning in games have been developed. In this section we present a brief overview of the field, and only go into details of the techniques that are directly relevant for this thesis. For a more extensive overview we refer to [76].

Many ideas for learning in games such as *rote learning*, *reinforcement learning*, and *comparison training*, as well as several searching techniques were introduced in Samuel's pioneering work [153, 154]. Nowadays, learning techniques are used for various aspects of game playing such as the opening book [35, 89, 116], search decisions [22, 51, 106, 121], evaluation functions [9, 11, 68, 72, 159, 169, 170], patterns and plans [38, 39, 109], and opponent models [18, 37, 60].

In this thesis we are interested in search decisions and evaluation functions. For search decisions we focus mainly on move ordering. For evaluation functions our focus is on various skills that are important for a Go player, such as scoring, predicting life and death, and estimating territory. To learn such skills two important learning paradigms are considered: (1) supervised learning, and (2) reinforcement learning, which will be discussed in subsections 6.2.1 and 6.2.2, respectively. For supervised learning we can use either classifiers from statistical pattern recognition or artificial neural networks (to be discussed in subsections 6.2.3 and 6.2.4). For reinforcement learning we only use artificial neural networks.

When we talk about learning or training we refer to the process of optimising an input/output mapping using mathematical techniques for function

approximation. Throughout this thesis the inputs are fixed-length feature vectors representing properties that are relevant for the given task. The outputs are usually scalars, but may in principle also be fixed-length vectors (for example to assign probabilities for multiple classes).

6.2.1 Supervised learning

Supervised learning is the process of learning from labelled examples. In supervised learning the training data typically consists of a large number of examples; for each example the input and the desired output are given. There are various supervised learning methods that can be used to optimise the input/output mapping such as Bayesian statistics [117], case-based reasoning [1], neural networks [19, 87], support vector machines [50, 123], and various other classifiers from statistical pattern recognition [93].

Although supervised learning is a powerful approach, it can only be used successfully when reliable training data is available. For Go sufficiently strong human players can, at least in principle, provide such training data. However, for many tasks this may become too time consuming. As an alternative it is, at least for some tasks, possible to use game records as a source of training data.

6.2.2 Reinforcement learning

When there is no reliable source of supervised training data, reinforcement learning [98, 168] can be used. Reinforcement learning is a technique for learning from delayed rewards (or punishments). In game playing such rewards are typically obtained at the end of the game based on the result. For improving the quality of play, the learning algorithm then has to distribute this reward over all state evaluations or actions that contributed to the outcome of the game. The most popular technique for learning state evaluations from delayed rewards is Temporal-Difference learning (TD) [167], which comes in many flavours [7, 9, 10, 30, 59, 68, 194]. A variation of TD-learning which, instead of learning to evaluate states, directly learns to evaluate actions is called Q-learning [138, 177, 195]. In games, Q-learning can for instance be used to learn to evaluate the moves directly without search. In contrast, standard TD-learning is typically used to learn to evaluate positions, which then require at least a one-ply search for an evaluation of the moves.

In introductory texts TD-learning and Q-learning are often applied in small domains where all possible states or state-action pairs can be stored in a lookup table. For most interesting tasks, however, lookup tables are not an option, at least for the following two reasons: (1) the state space is generally too large to store in memory, and (2) there is no generalisation between states. To overcome both problems the lookup table can be replaced by a function approximator, which in combination with a well-chosen representation may provide generalisation to unseen states. Combining TD-learning with function approximation is a non-trivial task that can lead to difficulties, especially when non-linear function approximators are used [7, 173].

So far, TD-learning has been reasonably successful in game playing, most notably by Tesauro's result in Backgammon [170]. In Go it has been applied by several researchers with some interesting results [51, 68, 72, 159, 202]. However, there are various alternative learning techniques that can be applied to the same tasks. Such techniques include Genetic Algorithms [24], Genetic Programming [49, 111], and some hybrid approaches [122, 201], which in recent years have gained quite some popularity in the field of game-playing [23, 48, 110, 142, 145, 150, 166]. Although these techniques have not yet produced world-class game-playing programs, they are certainly worth further investigation. However, they are beyond the scope of this thesis.

6.2.3 Classifiers from statistical pattern recognition

In this thesis we use a number of standard classifiers from statistical pattern recognition which are briefly discussed below. Most of the experiments with these classifiers were performed in MATLAB using PRTOOLS3 [65]. For an extensive overview of these and several other classifiers as well as a good introduction to the field of statistical pattern recognition we refer to [63, 93].

Before we discuss the classifiers it is important to note that all classifiers require a reasonably continuous feature space in which the compactness hypothesis holds. The compactness hypothesis states that "Representations of real world similar objects are close. There is no ground for any generalisation (induction) on representations that do not obey this demand." [5, 64]. The reason why compactness is important is that all classifiers use distance measures (usually Euclidean-like) to indicate similarity. Once these distance measures lose meaning the classifiers lose their ability to generalise to unseen instances.

The nearest mean classifier

The nearest mean classifier (NMC) is one of the simplest classifiers used for pattern recognition. It only stores the mean for each class, based on the training data, and assigns unseen instances to the class with the nearest mean.

The linear discriminant classifier

The linear discriminant classifier (LDC) computes the linear discriminant between the classes in the training data. The classifier approximates the optimal Bayes classifiers for classes with normal densities and equal covariance matrices.

The logistic linear classifier

The logistic linear classifier (LOGLC) is a linear classifier that maximises the likelihood criterion using the logistic (sigmoid) function. It is functionally equivalent to a perceptron network without hidden layers.

The quadratic discriminant classifier

The quadratic discriminant classifier (QDC) computes the quadratic discriminant between the classes in the training data. The classifier approximates the optimal Bayes classifiers for classes with normal densities.

The nearest neighbour classifier

The nearest neighbour classifier (NNC) is a conceptually straightforward classifier which stores all training data and assigns new instances to the class of the nearest example in the training set. For overlapping class distributions NNC behaves as a proportional classifier.

The k-nearest neighbours classifier

The k-nearest neighbours classifier (KNNC) is an extension of NNC, which stores all training examples. New instances are classified by assigning a class label based on the k nearest examples in the training set. Unlike NNC, which requires no training except storing all data, the KNNC has to be trained to find an optimal value for k. This is typically done by minimising the leave-one-out error on the training data. When the number of training examples becomes large KNNC approximates the optimal Bayes classifier.

6.2.4 Artificial neural networks

In the last decades there has been extensive research on artificial neural networks [19, 63, 80, 87]. Various types of network architectures have been investigated such as single-layer and multi-layer perceptron networks [19, 120, 148], simple recurrent networks [69], radial basis networks [46], networks of spiking neurons [78], as well as several closely related techniques such as Gaussian mixture models [119], self-organising maps [108], and support vector machines [50, 123].

Our main focus is on the well-known multi-layer perceptron (MLP) networks. The most common architecture for MLPs is feed-forward, where the input signals are propagated through one or more hidden layers with sigmoidal transfer functions to provide a non-linear mapping to the output. Simple feed-forward MLPs do not have the ability to learn sequential tasks where memory is required. Often this problem can be avoided by providing the networks with an extended input so that sequential structure can be learned directly. However, when this is not an option, the networks may be enhanced with feedback connections [69], or more specialised memory architectures [77, 88].

Once an adequate network architecture is selected it has to be trained to optimise the input/output mapping. The most successful training algorithms, at least for supervised learning tasks, are gradient based. For training the network the gradient consists of the partial derivatives for each network weight with respect to some error measure between the actual output values and the desired output values of the network. Usually the mean-square error is used, however other differentiable error measures can often be applied as well. For training

feed-forward networks the gradient can be calculated efficiently by repeated application of the chain rule, which is generally referred to as backpropagation [149]. For more complex recurrent networks several techniques can be applied for calculating the gradient [137]. Probably the most straightforward approach is backpropagation through time [179], which corresponds to performing standard backpropagation on the network unfolded in time.

Once the gradient information is available a method has to be selected for updating the network weights. There are many possible algorithms [81]. The algorithms typically aim at a good performance, speed, generalisation, and preventing premature convergence into local optima by using various tricks such as adding momentum, adaptive learning rate(s), batch learning, early stopping, line searches, random restarts, approximation of the Hessian matrix, and other heuristic techniques. In this thesis we use gradient descent with momentum and adaptive learning [81], and the resilient propagation algorithm (RPROP) [146]. We also did some preliminary experiments with Levenberg-Marquardt [82], quasi-Newton [58], and several conjugate gradient algorithms [81]. However, especially for large problems, these algorithms usually trained significantly slower or obtained less generalisation to unseen instances.

In this thesis artificial neural networks are used for evaluation tasks as well as for classification tasks. For evaluation tasks we typically use the network's continuous valued output(s) directly. For classification tasks an additional step is taken because a class has to be selected. In the simplest case of only two possible classes (such as yes or no) we typically use a network with one output and set a threshold to decide the class. In the case of multiple classes we normally use one output for each possible class. For unseen instances the class label is then typically selected by the output that has the highest value.

6.3 Fundamental questions

Our second research question is to what extent learning techniques can be used in computer Go. From the overview presented above it is clear that there are many interesting learning techniques which can be applied to the various aspects of game playing. Since it is impossible to investigate them all within the scope of one thesis we restricted our focus on artificial neural networks (ANNs) for learning search decisions and evaluation functions.

Out of the many types of ANNs we decided to restrict ourselves even further by focusing only on MLPs. However even for MLPs there are several possible architectures. A first question is the choice of the network architecture and, of course directly related to it, the choice of the representation. To choose the right network architecture and representation it is important to have some understanding of the strengths and weaknesses, as well as the fundamental limitations of the various alternatives. A second question is whether to use supervised learning or reinforcement learning.

To obtain some insight into the strengths and weaknesses of the various architectures and learning paradigms we decided to try our ideas on the simplified

domain of connectedness, which will be discussed in the next section. Then in the following chapters we will focus on two learning tasks for Go: (1) evaluating moves, and (2) evaluating positions. We will present supervised learning techniques for training feed-forward MLP architectures on both tasks, and compare them to standard classifiers from statistical pattern recognition.

6.4 Learning connectedness

The limitations of single-layer perceptrons were investigated by Minsky and Papert [120]. In 1969, their proof that single-layer perceptrons could not learn connectedness, as well as their (incorrect) assessment that the same would be true for multi-layer perceptrons (which they even repeated in the 1988 epilogue of the expanded edition), stifled research in most of the field for at least a decade. Nevertheless, their work is an interesting starting point for investigating the properties of MLPs. Although we now know that MLPs, with a sufficiently large number of hidden units, can approximate any function arbitrarily close this does not mean that practical learning algorithms are necessarily actually able to do this. Moreover, connectedness is still among the more difficult learning tasks, especially when regarding generalisation to unseen instances.

In Go connectedness is a fundamental element of the game because connections form blocks and chains, and connectivity is essential for recognising liberties as well as various other more subtle elements of the game that may be relevant, e.g., for spatial reasoning [25]. It is important that our learning system is able to handle these elements of the game. Consequently there are two important design choices. First, the architecture of the network has to be chosen. Here we have choices such as the number of neurons, hidden layers, and whether recurrent connections or specialised memory architectures are needed. Second, an adequate representation has to be chosen. Adequate representations can improve the performance and may avoid the need for complex network architectures because they facilitate generalisation and because they can implicitly perform some of the necessary computations more efficiently outside of the network.

We decided to test a number of different network architectures and learning algorithms on the task of learning to determine connectedness between stones from example positions. Note that we do not consider the possibility of connecting under alternating play, we just focus on the question whether a practical learning system can learn to detect that two stones are connected regardless of any additional moves. To make the task interesting we only used a direct representation of the raw board (so no additional features were calculated). Of course, a more complex representation could have solved the problem by providing the answer as an input feature. However, this would not give us any insight into the network's capabilities to learn such a feature from examples, which on a slightly more subtle level of connectedness may still be necessary at some point.

6.4.1 The network architectures

The standard feed-forward multi-layer perceptron architecture (MLP) for pattern classification usually has one hidden layer with non-linear transfer functions, is fully connected to all inputs, and has an output layer with one neuron assigned to each class. The disadvantage of using the MLP (or any other standard classifier) for raw board classification is that the architecture does not exploit any knowledge about the topological ordering of the intersections on the board. Although the intersections are topologically fixed on the rectangular grid, the conventional network architectures treat every intersection just as an (arbitrary) element of the input vector, thus ignoring the spatial order of the original representation. For humans this disadvantage becomes evident in the task of recognising natural images in which the spatial order of pixels is removed either by random permutation or by concatenation into a linear array. Clearly, for methods dealing with low-level image properties, the topological ordering is relevant. This observation motivated us to test a special input for our network architecture.

Inspired by the unrivalled performance of human vision and the fact that humans (and many other animals) have eyes we designed ERNA, an Eye-based Recurrent Network Architecture. Figure 6.1 shows the main components of ERNA. In our architecture, the eye is an input structure covering a local subset of intersections surrounding a movable point of fixation (see upper left corner). The focusing and scanning operations of the eye impose spatial order onto the input, thus automatically providing information about the topological ordering of the intersections.

The movement of the eye is controlled by five action neurons (left, right, up, down, stay). Together with the action neurons for classification (one for each class) they form the action layer (see upper right corner).

Focusing the eye on relevant intersections usually requires multiple actions. Since knowledge about previously observed pixels may be needed a memory seems necessary. It is implemented by adding recurrent connections to the network architecture. The simplest way to do this is linking the output of the hidden layer directly to the input. However, since information is partially redundant, an additional linear layer, called global memory, is applied to compress information between the output of the hidden layer and the input for the next iteration. (An interesting alternative would be to try LSTM instead [77, 88].)

Since the global memory has no topological ordering (with respect to the grid structure) and is overwritten at every iteration, it is not well suited for long-term storage of information related to specific locations on the board. Therefore, a local memory formed by linear neurons coupled to the position of the eye input is devised. At each iteration, the hidden layer is connected to the neurons of the local memory associated with the area visible by the eye. In ERNA the number of local memory neurons for an intersection as well as the readable and writable window size are defined in advance. The operation of the network is further facilitated by three extra input neurons representing the co-ordinates of the eye's point of fixation (X,Y) and the maximum number of iterations left (I).

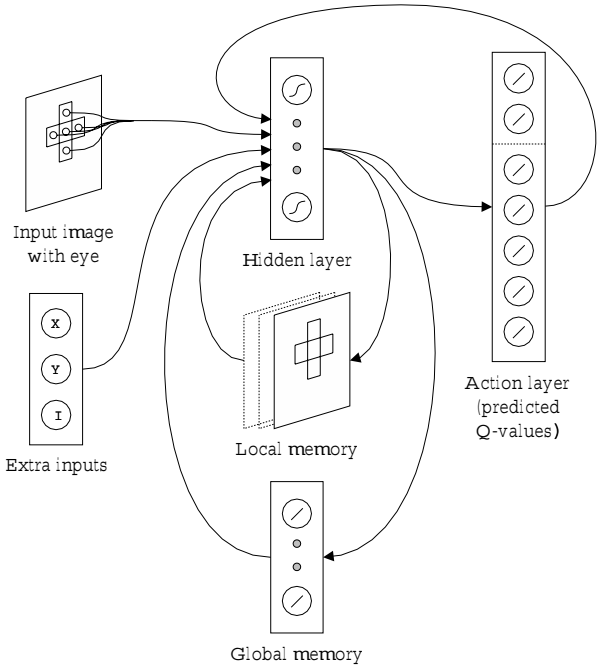


Figure 6.1: The ERNA architecture.

Below we briefly discuss the operation of ERNA. At each iteration step the hidden layer performs a non-linear mapping of input signals from the eye, the local memory, the global memory, the action layer and the three extra inputs to the local memory, the global memory and the action layer. The network then executes the action associated with the action neuron with the largest output value. The network iterates until the selected action performs the classification, or a maximum number of iterations is reached.

We note that, next to the normal recurrent connections of the memory, in ERNA the action layer is also recurrently connected to the hidden layer, thus allowing (back)propagation of information through all the action neurons.

Since the eye automatically incorporates knowledge about the topological ordering of intersections into the network architecture, we expect it to facilitate learning in topologically-oriented raw-board classification tasks, i.e., with the same number of training examples a better classification performance should be obtained. To evaluate the added value of the eye and that of the recurrent connections, ERNA is compared with three other network architectures.

The first network is the standard MLP, which has a feed-forward architecture with one non-linear hidden layer. The second network is a feed-forward network with an eye. This network is a stripped-down version of ERNA. All recurrent connections are removed by setting the number of neurons for local and global memory to zero. Previous action values are also not included in the input.

The third network is a recurrent network with a fully-connected input, a fully-connected recurrent hidden layer with non-linear transfer functions, and a linear output layer with an action neuron for each class and an extra action neuron for choosing another iteration (class thinking). The difference with the MLP is that the hidden layer has recurrent connections and the output layer has one more action neuron. This network architecture is very similar to the well-known Elman network [69] except that signals also propagate recurrently between the action layer and the hidden layer (as happens in ERNA).

6.4.2 The training procedure

In our experiments, we only used gradient-based learning techniques. The gradients, which are used to update the weights, consist of the partial derivatives of each network weight with respect to the (mean square) error between the actual output values and the target output values of the network. For the standard MLP the target values are directly available from the class information, as in supervised learning. For the other network architectures, which may select actions that do not directly lead to a classification, targets could not be calculated directly. Instead we used Q(λ)-learning [138, 167, 177]. For the feed-forward networks, without recurrent connections, we calculated the gradients using standard backpropagation. For the recurrent networks the gradients were calculated with backpropagation through time [179]. For updating the network weights we selected the resilient propagation algorithm (RPROP) developed by Riedmiller and Braun [146].

6.4.3 The data set

For the experiments, square 4×4, 5×5, and 6×6 board positions were created. Boards with the upper left stone connected to the lower right stone were labelled connected, all others were labelled unconnected. For simplicity we binarised the boards, thus treating enemy stones and free points equal (not connecting).

The boards were not generated completely at random, because on such data all networks would perform almost optimally. The reason is that in 75% of the cases the class unconnected can be determined from the two crucial corners alone (both must contain a stone for being connected), and in addition the number of placed stones is a strong indicator for connectedness.

We define a *minimal connected path* as a path of stones in which each stone is crucial for connectedness (if any stone is removed the two corners are no longer connected). To build a reasonably difficult data set, we started to generate the set of all minimal connected paths between the two corners. From this set a new set was generated by making copies and randomly flipping 15% of the points. For all boards both crucial corners always contained a stone. Duplicate boards and boards with less stones than the minimal path length (for connecting the two corners) were removed from the data set.

After applying this process for creating the 4×4, 5×5, and 6×6 boards, the three data sets were split into independent training and test sets, all containing

an equal number of unique positive and negative examples. The three sets contained 300, 1326, and 1826 training examples and 100, 440, and 608 test examples, respectively.

6.4.4 Experimental results

We compared the generalising ability of ERNA with the three other network architectures by focusing on the relation between the number of training examples and the classification performance on an independent test set. To prevent over-training, in each run a validation set was selected from the training examples and was used to estimate the optimal point for stopping the training. For the experiments with the 4×4 boards 100 validation samples were used. For both the 5×5 and 6×6 boards 200 validation samples were used.

The setting of ERNA

Because of limited computational resources and the fact that reinforcement learning is much slower than supervised learning, the size of the hidden layer was tested exhaustively only for the standard MLP. For ERNA we established reasonable settings, for the architecture and training parameters, based on some initial tests on 4×4 boards. Although these settings were kept the same for all our experiments, other settings might give better results especially for the larger boards. The architecture so obtained was as follows. For the hidden layer 25 neurons, with tangent sigmoid transfer functions, were used. The area observed by the eye contained the intersection on the fixation point and the four direct neighbours, i.e., the observed area was within a Manhattan distance of one from the centre point of focus. The output to the local memory was connected only to the centre point. For each point three linear neurons were assigned to the local memory. The global memory contained 15 linear neurons. All memory and action neurons were initialised at 0. During training, actions were selected randomly 5% of the time. In the rest of the cases, the best action was selected directly 75% of the time, and 25% of the time actions were selected with a probability proportional to their estimated action value. Of course, during validation and testing no exploration was used. The maximum number of iterations per example was set equal to the number of intersections. Negative reinforcements of -1 were returned for (1) moving the eye out of range, (2) exceeding the maximum number of iterations, and (3) performing the wrong classification. A positive reinforcement of $+1$ was returned for the correct classification. The Q-learning parameters λ and γ were set at 0.3 and 0.97. All network weights were initialised with small random values. Training was performed in batch for a maximum of 5000 epochs.

Settings of the three other architectures

The MLP was tested with hidden layers of 3, 6, 12, 25, 50, and 100 neurons. In each run, the optimal layer size was selected based on the performance on the

validation set. Supervised training with RPROP was performed in batch for a maximum of 2000 epochs.

The stripped-down version of ERNA (the feed-forward network with eye) was kept similar to ERNA as much as possible. The sizes of the hidden layer and the eye were kept the same and training was done with exactly the same learning parameters.

The fully-connected recurrent (Elman-like) network also used a hidden layer of 25 neurons, and training was done with exactly the same learning parameters except that this network was allowed to train for a maximum of 10,000 epochs.

The four architectures compared

In Figures 6.2, 6.3, and 6.4 the average performance is plotted for the four network architectures tested on the 4×4, 5×5 and 6×6 boards, respectively. The horizontal axis shows the number of training examples, with logarithmic scaling. The vertical axis shows the fraction of correctly-classified test samples (1.0 for perfect classification, 0.5 for pure guessing).

The plots show that for all board sizes both ERNA and the stripped-down version of ERNA outperform the two networks without eye. Moreover, we can see that the recurrent connections are only useful for ERNA, and then only when sufficient training examples are available.

We also compared the neural networks to some of the standard classifiers from statistical pattern recognition (see 6.2.3). Since these classifiers are not trained incrementally, unlike the neural networks, we combined the training and validation sets resulting in 300 examples for the 4×4 board, 1000 examples for the 5×5 board, and 1000 examples for the 6×6 board. In Table 6.1 the results are shown for NMC, LDC, QDC, NNC, KNNC, as well as the four network architectures. It is shown that the performance of the network architectures is quite acceptable compared to most standard classifiers. Moreover, also here we see that the specialised architectures outperform the other classifiers at least with respect to generalisation.

	Board size		
Classifier	4×4	5×5	6×6
Nearest mean	71%	61%	56%
Linear Discriminant	74%	64%	60%
Quadratic discriminant	78%	75%	76%
Nearest neighbour	67%	72%	58%
K-nearest neighbours	77%	74%	63%
Feed-forward MLP	80%	74%	67%
Fully recurrent MLP	83%	77%	67%
Feed-forward + eye (stripped down ERNA)	91%	86%	76%
Recurrent + eye (ERNA)	97%	95%	87%

Table 6.1: Comparison with some standard classifiers.

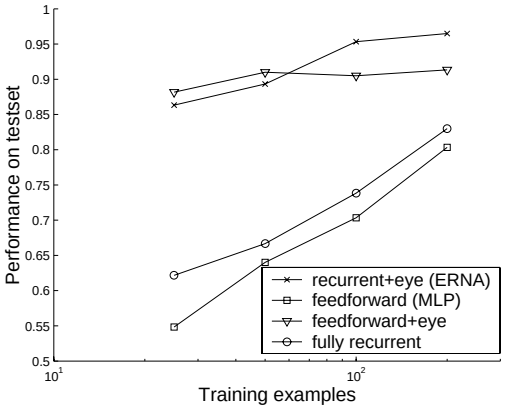


Figure 6.2: Connectedness on 4×4 boards.

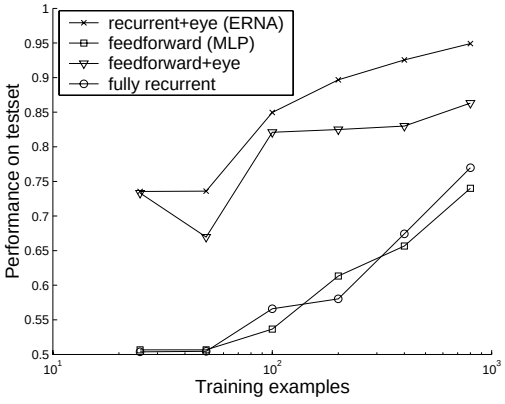


Figure 6.3: Connectedness on 5×5 boards.

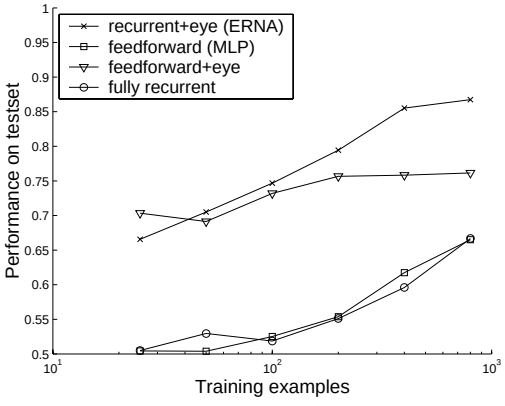


Figure 6.4: Connectedness on 6×6 boards.

6.4.5 Discussion

The experiments indicate that practical learning algorithms are able to learn connectedness between stones at least on relatively small boards. As the board size increases the number of instances that is required for sufficient generalisation also increases drastically. Although all the four architectures can learn to perform the same task, the number of training examples that is needed to obtain the same level of generalisation varies greatly. The experiments indicate that using specialised architectures that can focus on local regions of the board significantly improves generalisation.

A point that was not highlighted above is speed. Although the specialised recurrent architectures can improve generalisation, compared to, for example, the standard MLP, the computational costs for training and operating such architectures is much higher. There are at least two reasons for the higher costs. (1) Reinforcement learning methods like Q(λ)-learning converge much slower (although using RPROP was a quite substantial help here) than supervised learning methods, probably because of a variety of reasons, such as the necessary exploration of the state space, the length of the action sequences, and the fact that there may also be some instability due to the use of (non-linear) function approximators. (2) Even when the networks are fully trained, the operation generally requires several steps (with internal feedback) before a final action is selected. Although we did not optimise ERNA to the full extent possible, it is safe to conclude that it cannot operate at speeds comparable to simpler classifiers such as the standard MLP.

Although it is too early for definite conclusions, we can already say something about the questions posed in section 6.3 regarding architecture, representation, and learning paradigm. Since both training and operation of complex recurrent network architectures is slow, it is our opinion that such architectures should only be used for tasks where supervised learning with large numbers of labelled training examples is not an option. In Go there are many tasks for which sufficient training examples can be obtained without too much effort. Moreover, since there is extensive human knowledge about the basic topological properties that are relevant for assessing Go positions it may be best to provide simple architectures with a well-chosen representation, which may easily compensate their reduced generalising capabilities observed for raw-board representations. Consequently, in the next chapters, instead of using complex recurrent architectures trained with reinforcement learning, we will focus on supervised learning in combination with well-chosen representations to obtain both speed and generalisation.

Chapter 7

Move prediction

This chapter is based on E. C. D. van der Werf, J. W. H. M. Uiterwijk, E. O. Postma, and H. J. van den Herik. Local move prediction in Go. In J. Schaeffer, M. Müller, and Y. Björnsson, editors, *Computers and Games: Third International Conference, CG 2002, Edmonton, Canada, July 2002: revised papers*, volume 2883 of *LNCS*, pages 393–412. Springer-Verlag, Berlin, 2003.[1]

In this chapter we investigate methods for building a system that can learn to predict expert moves from examples. An important application for such predictions is the move ordering for $\alpha\beta$ tree search. Furthermore, good move predictions can also be used to reduce the number of moves that will be investigated (forward pruning), to bias a search towards more promising lines of play, and, at least in theory, to avoid all search completely.

It is known that many moves in Go conform to some local pattern of play which is performed almost reflexively by human players. The reflexive nature of many moves leads us to believe that pattern-recognition techniques, such as neural networks, are capable of predicting many of the moves made by human experts. The encouraging results reported by Enderton [70] and Dahl [51] on similar supervised-learning tasks, and by Schraudolph [159] who used neural networks in a reinforcement-learning framework, underline our belief.

Since locality seems to be important in Go, our primary focus is on the ranking that can be made between legal moves which are near to each other. Ideally, in a local ranking, the move played by the expert should be among the best. Of course, we are also interested in full-board ranking. However, due to the complexity of the game and the size of any reasonable feature space to describe adequately the full board, full-board ranking is not our main aim.

The remainder of this chapter is organised as follows. In section 7.1 the move predictor is introduced. In section 7.2 we discuss the representation that is used by the move predictor for ranking the moves. Section 7.3 presents feature-extraction and pre-scaling methods for extracting promising features to

[1] The author would like to thank Springer-Verlag and his co-authors for permission to reuse relevant parts of the article in this thesis.

reduce the dimensionality of the raw-feature space, and we discuss the option of a second-phase training. Then section 7.4 presents experimental results on the performance of the raw features, the feature extraction, the pre-scaling, and the second-phase training. From the experiments we select our best move predictor (MP*). In section 7.5 we assess the quality of MP* (1) by comparing it to the performance of human amateurs on a local prediction task, (2) by testing on professional games, and (3) by actually playing against the program GNU Go. Finally, section 7.6 provides conclusions and suggestions for future research.

7.1 The move predictor

The goal of the move predictor is to rank legal moves, based on a set of features, in such a way that expert moves are ranked above (most) other moves. In order to rank moves they must be made comparable. This can be done by performing a non-linear mapping of the feature vector onto a scalar value for each legal move. A general function approximator, such as a neural network, which can be trained from examples, can perform such a mapping.

The architecture chosen for our move predictor is the well-known feed-forward multi layer perceptron (MLP). Our MLP has one hidden layer with non-linear transfer functions, is fully connected to all inputs, and has a single linear output predicting the value for ranking the move. Although this move predictor alone may not suffice to play a strong game, e.g., because it may have difficulty to understand tactical threats that require a deep search, it can be of great value for move ordering and for reducing the number of moves that have to be considered globally.

Functionally the network architecture is identical to the half-networks used in Tesauro's comparison training [169], which were trained by standard back propagation with momentum. Our approach differs from Tesauro's in that it employs a more efficient training scheme and error function especially designed for the task of move ordering.

7.1.1 The training algorithm

The MLP must be trained in such a way that expert moves are generally valued higher than other moves. Several training algorithms exist that can be used for this task. In our experiments, the MLP was trained with the resilient propagation algorithm (RPROP) developed by Riedmiller and Braun [146]. RPROP is a gradient-based training procedure that overcomes the disadvantages of gradient-descent techniques (slowness, blurred adaptivity, tuning of learning parameters, etc.). The gradient used by RPROP consists of partial derivatives of each network weight with respect to the (mean-square) error between the actual output values and the desired output values of the network.

In standard pattern-classification tasks the desired output values are usually set to zero or one, depending on the class information. Although for the task of move prediction we also have some kind of class information (expert / non-

expert moves) a strict class assignment is not feasible because class membership may change during the game, i.e., moves which may initially be sub-optimal, later on in the game can become expert moves [70]. Another problem, related to the efficiency of fixed targets, is that when the classification or ordering is correct, i.e., the expert move is valued higher than the non-expert move(s), the network needlessly adapts its weights just to get closer to the target values.

To incorporate the relative nature of the move values into the training algorithm, training is performed with move pairs. A pair of moves is selected in such a way that one move is the expert move, and the other is a randomly selected move within a pre-defined maximum distance from the expert move.

With this pair of moves we can devise the following error function

$$E(v_e, v_r) = \begin{cases} (v_r + \epsilon - v_e)^2, & \forall\ v_r + \epsilon > v_e \\ 0, & otherwise \end{cases} \tag{7.1}$$

in which v_e is the predicted value for the expert move, v_r is the predicted value for the random move and ϵ is a control parameter that scales the desired minimal distance between the two moves. A positive value for control parameter ϵ is needed to rule out trivial solutions where all predicted values v would become equal. Although not explicitly formulated in his report [70] Enderton appears to use the same error function with $\epsilon = 0.2$.

Clearly the error function penalises situations where the expert move is ranked below the non-expert move. In the case of a correct ranking the error can become zero (just by increasing the scale), thus avoiding needless adaptations. The exact value of control parameter ϵ is not very important, as long as it does not interfere with minimum or maximum step-sizes for the weight updates. (Typical settings for ϵ were tested in the range $[0.1, 1]$ in combination with standard RPROP settings.)

Repeated application of the chain rule, using standard backpropagation, calculates the gradient from equation 7.1. A nice property of the error function is that no gradient needs to be calculated when the error signal is zero (which practically never happens for the standard fixed target approach). As the performance grows, this significantly reduces the time between weight updates.

The quality of the weight updates strongly depends on the generalisation of the calculated gradient. Therefore, all training is performed in batch, i.e., the gradient is averaged over all training examples before performing the RPROP weight update. To avoid overfitting, training is terminated when the performance on an independent validation set does not improve for a pre-defined number of weight updates. (In our experiments this number is set to 100.)

7.2 The representation

In this section we present a representation with a selection of features that can be used as inputs for the move predictor. The list is by no means complete and could be extended with a (possibly huge) number of Go-specific features. Our selection comprises a simple set of locally computable features that are

common in most Go representations [51, 70] used for similar learning tasks. In a tournament program our representation can readily be enriched with additional features which may be obtained by a more extensive (full-board) analysis or by specific goal-directed searches.

Our selection consists of the following eight features: stones, edge, ko, liberties, liberties after, captures, nearest stones, and the last move.

Stones

The most fundamental features to represent the board contain the positions of black and white stones. Local stone configurations can be represented by two bitmaps. One bitmap represents the positions of the black stones, the other represents the positions of the white stones.

Since locality is important, it seems natural to define a local region of interest (ROI) centred on the move under consideration. The points inside the ROI are then selected from the two bitmaps, and concatenated into one binary vector; points outside the ROI are discarded.

A question which arises when defining the ROI is: what are good shapes and sizes? To answer this question we tested differently sized shapes of square, diamond, and circular forms, as shown in Figure 7.1, all centred on the free point considered for play. For simplicity (and for saving training time) we only included the edge and the ko features. In Figure 7.2 the percentages of incorrectly ordered move-pairs are plotted for the ROIs. These results were obtained from several runs with training sets of 100,000 feature vectors. Performances were measured on independent test sets. The standard deviations (not shown) were less than 0.5%.

The results do not reveal significant differences between the three shapes. In contrast, the size of the ROI does affect the performance considerably. Figure 7.2 clearly shows that initially, as the size of the ROI grows, the error decreases. However, at a size of about 30 the error starts to increase with the size. The increase of the classification error with the size (or dimensionality) of the input is known as the "peaking phenomenon" [92] which is caused by the "curse of dimensionality" [13] to be discussed in section 7.3.

Since the shape of the ROI is not critical for the performance, in all further experiments we employ a fixed shape, i.e., the diamond. The optimal size of the ROI cannot yet be determined at this point since it depends on other factors such as the number of training patterns, the number of other features, and the performance of the feature-extraction methods that will be discussed in the next section.

Edge

In Go, the edge has a huge impact on the game. The edge can be encoded in two ways: (1) by including the coordinates of the move that is considered for play, or (2) by a binary representation (board = 0, edge = 1) using a 9-bit string vector along the horizontal and the vertical line from the move towards the closest

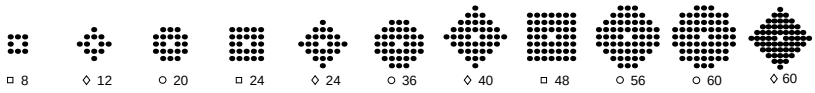


Figure 7.1: Shapes and sizes of the ROI.

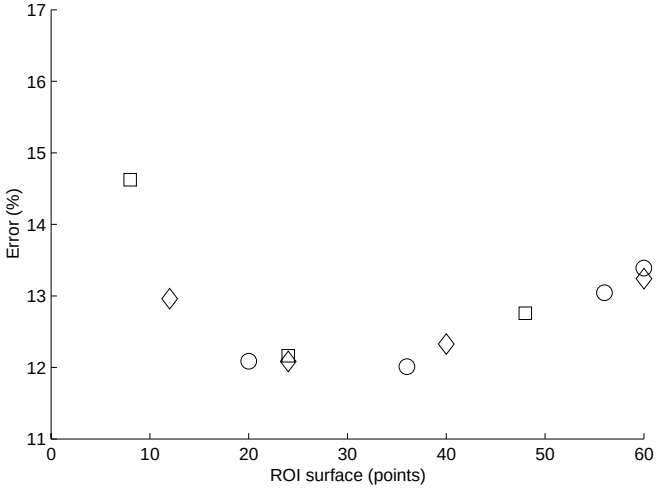


Figure 7.2: Performance for different ROIs.

edges. Preliminary experiments showed a slightly better performance for the binary representation (around 0.75%). Therefore we implemented the binary representation. (Note that since the integer representation can be obtained by a linear combination of the larger binary representation, there is no need to implement both. Furthermore, since on square boards the binary representation can directly be transformed to a full binary ROI by a logical OR it is not useful to include more edge features.)

Ko

The ko rule, which forbids returning to previous board states, has a significant impact on the game. In a ko fight the ko rule forces players to play threatening moves, elsewhere on the board, which have to be answered locally by the opponent. Such moves often are only good in a ko fight since they otherwise just reduce the player's number of ko threats. For the experiments presented here only two ko features were included. The first one is a binary feature that indicates if there is a ko or not. The second one is the distance from the point considered for play to the point that is illegal due to the ko rule. In a tournament program it may be wise to include more information, such as (an estimate of) the value of the ko (number of points associated with winning or losing the ko), assuming this information is available.

Liberties

An important feature used by human players is the number of liberties. The number of liberties is the number of unique free intersections connected to a stone. The number of liberties of a stone is a lower bound on the number of moves that must be played by the opponent to capture that stone.

In our implementation for each stone inside a diamond-shaped ROI the number of liberties is calculated. For each empty intersection only the number of directly neighbouring free intersections is calculated.

Liberties after

The feature 'liberties after' is directly related to the previous one. It is the number of liberties that a new stone will have after placement on the position that is considered. The same feature is also calculated for the opponent moving first on this position.

Captures

It is important to know how many stones are captured when Black or White plays a stone on the point under investigation. Therefore we include two features. The first feature is the number of stones that are directly captured (removed from the board) after placing a (friendly) stone on the intersection considered. The second feature is the number of captures if the opponent would move first on that intersection.

Nearest stones

In Go stones can have long-range effects which are not visible inside the ROI. To detect such long-range effects a number of features are incorporated characterising stones outside the ROI.

Since we do not want to incorporate all stones outside the ROI, features are calculated only for a limited set of stones near the point considered for play. These nearest stones are found by searching in eight directions (2 horizontal, 2 vertical and 4 diagonal) starting from the points just outside the ROI. In Figure 7.3 the two principal orderings for these searches are shown. The marked stone, directly surrounded by the diamond-shaped ROI, represents the point considered for play. Outside the ROI a horizontal beam and a diagonal beam of numbered stones are shown representing the order in which stones are searched. For each stone found we store the following four features: (1) colour, (2) Manhattan distance to the proposed move, (3) offset perpendicular to the beam direction, and (4) number of liberties.

In our implementation only the first stone found in each direction is used. However, at least in principle, a more detailed set of features might improve the performance. This can be done by searching for more than one stone per direction or by using more (narrow) beams (at the cost of increased dimensionality).

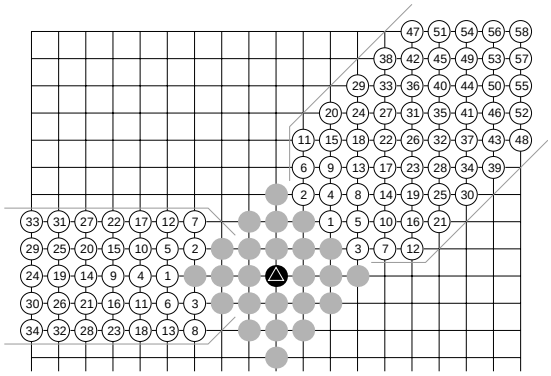


Figure 7.3: Ordering of nearest stones.

Last move

The last feature in our representation is the Manhattan distance to the last move played by the opponent. This feature is often a powerful cue to know where the action is. In the experiments performed by Enderton [70] (where bad moves were selected randomly from all legal moves) this feature was even considered harmful since it dominated all others and made the program play all moves directly adjacent to the last move played. However, since in our experiments both moves are selected from a local region we do not expect such a dramatic result.

Exploiting symmetry

A standard technique to simplify the learning task, which is applied before calculating the features, is canonical ordering and colour reversal. The game of Go is played on a square board which contains eight symmetries. Furthermore, if we ignore the komi, positions with Black to move are equal to positions where White is to move if all stones reverse colour. Rotating the viewpoint of the move under consideration to a canonical region in one corner, and reversing colours so that only one side is always to move, effectively reduces the state space by a factor approaching 16.

7.3 Feature extraction and pre-scaling

The representation, presented in the previous section, can grow quite large as more information is added (either by simply increasing the ROIs or by adding extra Go-specific knowledge). The high-dimensional feature vectors significantly slow down training and can even decrease the performance (see Figure 7.2). As stated in section 7.2 the negative effect on performance is known as the "curse of dimensionality" [13] and refers to the exponential growth of the hyper-volume of the feature space as a function of its dimensionality. The "curse of

dimensionality" leads to the paradoxical "peaking phenomenon" where adding more information decreases classification performance [19, 92]. Efficiency and performance may be reduced even further due to the fact that some of the features are correlated or redundant.

In this section we discuss the following topics: feature-extraction methods (in 7.3.1), pre-scaling the raw feature vector (in 7.3.2), and second-phase training (in 7.3.3).

7.3.1 Feature-extraction methods

Feature-extraction methods deal with peaking phenomena by reducing the dimensionality of the raw-feature space. In pattern recognition there is a wide range of methods for feature extraction such as principal component analysis [96], discriminant analysis [75], independent component analysis [99], Kohonen's mapping [108], Sammon's mapping [152] and auto-associative diabolo networks [180]. In this section we will focus on simple linear feature-extraction methods, which can efficiently be used in combination with feed-forward networks.

Principal component analysis

The most well known feature-extraction method is principal component analysis (PCA). PCA is an unsupervised feature-extraction method that approximates the data by a linear subspace using the mean-square-error criterion. Mathematically, PCA finds an orthonormal linear projection that maximises preserved variance. Since the amount of variance is equal to the trace of the covariance matrix, an orthonormal projection that maximises the trace for the extracted feature space is considered optimal. Such a projection is constructed by selecting the eigenvectors with the largest eigenvalues of the covariance matrix. The PCA technique is of main importance since all other feature-extraction methods discussed here rely on trace maximisation. The only difference is that for other mappings other (covariance-like) matrices are used.

Discriminant analysis

PCA works well in a large number of domains where there is no class information available. However, in domains where class information is available supervised feature-extraction methods usually work better. The task of learning to predict moves from examples is a supervised-learning task. A commonly used supervised feature-extraction method is linear discriminant analysis (LDA). In LDA, inter-class separation is emphasised by replacing the covariance matrix in PCA by a general separability measure known as the Fisher criterion, which results in finding the eigenvectors of $S_w^{-1}S_b$ (the product of the inverse of the within-class scatter matrix, S_w, and the between-class scatter matrix S_b) [75].

The Fisher criterion is known to be optimal for linearly separable Gaussian class distributions only. However, in the case of move prediction, the two classes of random and expert moves are not likely to be so easily separable for at least

two reasons. First, the random moves can become expert moves later on in the game. Second, sometimes the random moves may even be just as good as (or better than) the expert moves. Therefore, standard LDA may not be well suited for the job.

Move-pair scatter

As an alternative to LDA we propose a new measure, the move-pair scatter, which may be better suited to emphasise the differences between good and bad moves. The move-pair scatter matrix S_{mp} is given by

$$S_{mp} = E[(x_e - x_r)(x_e - x_r)^T] \tag{7.2}$$

in which E is the expectation operator, x_e is the expert vector and x_r the associated random vector. It should be noted that the trace of S_{mp} is the expected quadratic distance between a pair of vectors.

The move-pair scatter matrix S_{mp} can be used to replace the between-class scatter matrix in the Fisher criterion. Alternatively a mapping that directly maximises move-pair separation is obtained by replacing the covariance matrix in PCA by S_{mp}. We call this mapping, on the largest eigenvectors of S_{mp}, move pair analysis (MPA).

Although MPA linearly maximises the preserved distances between the move pairs, which is generally a good idea for separability, it has one serious flaw. Since the mapping aggressively tries to extract features which separate the move pairs, it can miss some features which are also relevant but have a more global and static nature. An example is the binary ko feature. In Go the ko rule can significantly alter the ordering of moves, i.e., a move which is good in a ko fight can be bad in a position without a ko. However, since this ko feature is globally set for both expert and random moves the preservable distance will always be zero, and thus the feature is regarded as uninteresting.

To overcome this problem the mapping has to be balanced with a mapping that preserves global structure such as PCA. This can be done by extracting a set of features, preserving global structure, using standard PCA followed by extracting a set of features using MPA on the subspace orthogonal to the PCA features. In the experimental section we will refer to this type of balanced feature extraction as PCAMPA. Naturally balanced extraction can also be performed in reversed order starting with MPA, followed by PCA performed on the subspace orthogonal to the MPA features, to which we will refer as MPAPCA. Another approach, the construction of a balanced scatter matrix by averaging a weighted sum of scatter and covariance matrices, will not be explored in this thesis.

Eigenspace separation transforms

Recently an interesting supervised feature-extraction method, called the eigenspace separation transform (EST), was proposed by Torrieri [171]. EST aims at maximising the difference in average length of vectors in two classes, measured

by the absolute value of the trace of the correlation difference matrix. For the task of move prediction the correlation difference matrix M is defined by

$$M = E[x_e x_e^T] - E[x_r x_r^T] \tag{7.3}$$

subtracting the correlation matrix of random move vectors from the correlation matrix of expert move vectors. (Notice that the trace of M is the expected quadratic length of expert vectors minus the expected quadratic length of random vectors.) From M we calculate the eigenvectors and eigenvalues. If the sum of positive eigenvalues is larger than the absolute sum of negative eigenvalues, the positive eigenvectors are used for the projection. Otherwise, if the absolute sum of negative eigenvalues is larger, the negative eigenvectors are used. In the unlikely event of an equal sum, the smaller set of eigenvectors is selected.

Effectively EST tries to project one class close to the origin while having the other as far away as possible. The choice for either the positive or the negative eigenvectors is directly related to the choice which of the two classes will be close to the origin and which will be far away. Torrieri [171] experimentally showed that EST performed well in combination with radial basis networks on an outlier-sensitive classification task. However in general the choice for only positive or negative eigenvectors seems questionable.

In principle it should not matter which of the two classes are close to the origin, along a certain axis, as long as the classes are well separable. Therefore, we modify the EST by taking eigenvectors with large eigenvalues regardless of their sign. This feature-extraction method, which we call modified eigenspace separation transform (MEST), is easily implemented by taking the absolute value of the eigenvalues before ordering the eigenvectors of M.

7.3.2 Pre-scaling the raw feature vector

Just like standard PCA, all feature-extraction methods discussed here (except LDA) are sensitive to scaling of the raw input features. Therefore features have to be scaled to an appropriate range. The standard solution for this is to subtract the mean and divide by the standard deviation for each individual feature. Another simple solution is to scale the minimum and maximum values of the features to a fixed range. The latter method however can give bad results in the case of outliers. In the absence of a priori knowledge of the data, such uninformed scalings usually work well. However, for the task of move prediction, we have knowledge on what our raw features represent and what features are likely to be important. Therefore we may be able to use this extra knowledge for finding an informed scaling that emphasises relevant information.

Since moves are more likely to be influenced by stones that are close than stones that are far away, it is often good to preserve most information from the central region of the ROI. This is done by scaling the variance of the individual features of the ROI inversely proportional to the distance to the centre point. In combination with the previously described feature-extraction methods this biases our extracted features towards representing local differences, while still keeping a relatively large field of view.

7.3.3 Second-phase training

A direct consequence of applying the feature-extraction methods discussed above is that potentially important information may not be available for training the move predictor. Although the associated loss in performance may be compensated by the gain in performance resulting from the reduced dimensionality, it can imply that we have not used all raw features to their full potential. Fortunately, there is a simple way to have the best of both worlds by making the full representation available to the network, and improve the performance further.

Since we used a linear feature extractor, the mapping to extract features from the original representation is a linear mapping. In the move predictor, the mapping from input to hidden layer is linear too. As a consequence both mappings can be combined into one (simply by multiplying the matrices). The result is a network that takes the features of the full representation as input, with the performance obtained on the extracted features.

Second-phase training entails the (re)training of the network formed by the combined mapping. Since the full representation is now directly available, the extra free parameters (i.e.,weights) give way for further improvement. (Naturally the validation set prevents the performance from getting worse.)

7.4 Experimental results

In this section we present experimental results obtained with our approach on predicting the moves of strong human players. Most games used for the experiments presented here were played on IGS [90]. Only games from rated players were used. Although we mainly used dan-level games, a small number of games played by low(er)-ranked players was incorporated in the training set too. The reason was our belief that the network should be exposed to positions somewhat less regular, which are likely to appear in the games of weaker players.

The raw feature vectors for the move pairs were obtained by replaying the games, selecting for each move the actual move that was played by the expert together with a second move selected randomly from all other free positions within a Manhattan distance of 3 from the expert move.

The dataset was split up into three subsets. One for training, one for validation (deciding when to stop training), and one for testing. Due to time constraints most experiments were performed with a relatively small data set. Unless stated otherwise, the training set contained 25,000 examples (12,500 move pairs); the validation and test set contained 5,000 and 20,000 (independent) examples, respectively. The predictor had one hidden layer of 100 neurons with hyperbolic tangent transfer functions.

The rest of this section is organised as follows. In subsection 7.4.1 we investigate the relative contribution of individual feature types. In subsection 7.4.2 we present experimental results for different feature-extraction and prescaling methods. In subsection 7.4.3 we show the gain in performance by the second-phase training.

7.4.1 Relative contribution of individual feature types

A strong predictor requires good features. Therefore, an important question for building a strong predictor is: how good are the features? The answer to this question can be found experimentally by measuring the performance of different configurations of feature types. The results can be influenced by the number of examples, peaking phenomena, and a possible bias of the predictor towards certain distributions. Although our experiments are not exhaustive, they give a reasonable indication of the relative contribution of the feature types.

We performed two experiments. In the first experiment we trained the predictor with only one feature type as input. Naturally, the performance of most single feature types is not expected to be high. The added performance (compared to 50% for pure guessing) is shown in the column headed "Individual" of Table 7.1. The first experiment shows that individually, the stones, liberties and the last move are the strongest features.

	Individual (%)	Leave one out (%)
Stones	+32.9	−4.8
Edge	+9.5	−0.9
Ko	+0.3	−0.1
Liberties	+24.8	0.0
Liberties after	+12.1	−0.1
Captures	+5.6	−0.8
Nearest stones	+6.2	+0.3
Last move	+21.5	−0.8

Table 7.1: Added performance in percents of raw-feature types.

For the second experiment, we trained the predictor on all feature types except one, and compared the performance to a predictor using all feature types. The results, shown in the last column of Table 7.1, indicate that again the stones are the best feature type. (It should be noted that negative values indicate good performance of the feature type that is left out.) The edge, captures and the last move also yield a small gain in performance. For the other features there seems to be a fair degree of redundancy, and possibly some of them are better left out. However, it may be that the added value of these features is only in the combination with other features. The liberties might benefit from reducing the size of their ROI. The nearest stones performed poorly (4 out of 5 times), which resulted in an average increase in performance of 0.3% after leaving these features out, possibly due to peaking effects. However, the standard deviations, which were around 0.5%, do not allow strong conclusions.

7.4.2 Performance of feature extraction and pre-scaling

Feature extraction reduces the dimensionality, while preserving relevant information, to overcome harmful effects related to the curse of dimensionality. In

section 7.3 a number of methods for feature extraction and pre-scaling of the data were discussed. Here we present empirical results on the performance of the feature-extraction methods discussed in combination with the three techniques for pre-scaling the raw feature vectors, discussed in subsection 7.3.2.

Table 7.2 lists the results for the different feature-extraction and pre-scaling methods. In the first row the pre-scaling is shown. The three types of pre-scaling are (from left to right): (1) normalised mean and standard deviation ($[\mu, \sigma]$), (2) fixed-range pre-scaling ($[\min, \max]$), and (3) ROI-scaled mean and standard deviation ($[\mu, \sigma]$, $\sigma^2 \sim 1/d$ in ROI). The second row shows the (reduced) dimensionality, as a percentage of the dimensionality of the original raw-feature space. Rows 3 to 11 show the percentages of correctly ranked move pairs, measured on the independent test set, for the nine different feature-extraction methods. Though the performances shown are averages of only a small number of experiments, all standard deviations were less than 1%. It should be noted that LDA* was obtained by replacing S_b with S_{mp}. Both LDA and LDA* used a small regularisation term (to avoid invertability and singularity problems). The balanced mappings, PCAMPA and MPAPCA, both used 50% PCA and 50% MPA.

pre-scaling	$[\mu, \sigma]$		$[\min, \max]$		$[\mu, \sigma]$, $\sigma^2 \sim 1/d$ in ROI			
Dimensionality	10%	25%	10%	25%	10%	25%	50%	90%
PCA	78.7	80.8	74.4	80.4	83.9	85.8	84.9	**84.5**
LDA	75.2	74.9	75.2	75.5	75.3	75.4	75.8	76.9
LDA*	72.5	73.8	70.0	72.1	72.6	74.0	75.9	76.7
MPA	73.7	76.7	70.3	73.8	80.7	84.6	**85.5**	84.3
PCAMPA	77.3	80.6	73.5	80.5	84.4	**85.9**	84.1	83.7
MPAPCA	77.2	80.6	74.0	80.1	83.6	85.6	84.4	84.3
EST	80.7	79.3	78.1	78.9	83.5	81.1	79.2	79.5
MEST	**84.3**	82.4	**82.6**	**82.2**	86.0	84.6	83.9	84.4
MESTMPA	83.8	**82.5**	**82.6**	**82.2**	**86.8**	85.6	84.5	**84.5**

Table 7.2: Performance in percents of extractors for different dimensionalities.

Table 7.2 reveals seven important findings.

- A priori knowledge of the feature space is useful for pre-scaling the data as is evident from the overall higher scores obtained with the scaled ROIs.
- In the absence of a priori knowledge it is better to scale by the mean and standard deviation than by the minimum and maximum values as follows from a comparison of the results for both pre-scaling methods.
- PCA performs quite well despite the fact that it does not use class information.
- LDA performs poorly. Replacing the between-class scatter matrix in LDA with our move-pair scatter matrix (i.e., LDA*) degrades rather than upgrades the performance. We suspect that the main reason for this is that

minimisation of the within-class scatter matrix, which is very similar to the successful covariance matrix used by PCA, is extremely harmful to the performance.

- MPA performs reasonably well, but is inferior to PCA. Presumably this is due to the level of global information in MPA.
- The balanced mappings PCAMPA and MPAPCA are competitive and sometimes even outperform PCA.
- MEST is the clear winner. It outperforms both PCA and the balanced mappings.

Our modification of the eigenspace separation transform (MEST) significantly outperforms standard EST. However, MEST does not seem to be very effective at the higher dimensionalities. It may therefore be useful to balance this mapping with one or possibly two other mappings such as MPA or PCA. One equally balanced combination of MEST and MPA is shown in the last row, other possible combinations are left for future study.

7.4.3 Second-phase training

After training of the move predictor on the extracted features we turn to the second-phase training. Table 7.3 displays the performances for both training phases performed on a training set of 200,000 examples. The first column (ROI) shows the size in number of intersections of the ROIs (a, b), in which a refers to the ROI for the stones and b refers to the ROI for the liberties. In the second column the dimensionality of the extracted feature space is shown, as a percentage of the original feature space. The rest of the table shows the performances and duration of the first-phase and second-phase training experiments.

As is evident from the results in Table 7.3, second-phase training boosts the performance obtained with the first-phase training. The extra performance comes at the price of increased training time, though it should be noted that in these experiments up to 60% of the time was spent on the stopping criterion, which can be reduced by setting a lower threshold.

	Phase 1			Phase 2	
ROI	dim. (%)	perf.(%)	time (h)	perf.(%)	time (h)
40,40	10	87.3	14.9	89.9	33.0
40,40	15	88.8	12.0	90.5	35.9
40,12	20	88.4	11.6	90.1	18.5
60,60	15	89.0	10.2	90.4	42.5
60,24	20	89.2	16.9	90.7	31.0
60,12	20	88.8	11.8	90.2	26.1

Table 7.3: First-phase and second-phase training statistics.

7.5 Assessing the quality of the move predictor

Below we assess the quality of our best move predictor (MP*). It is the network with an ROI of size 60 for the stones and 24 for the liberties, used in subsection 7.4.3; it is prepared by ROI-scaled pre-scaling, MESTMPA feature extraction, first-phase and second-phase training. In subsection 7.5.1 we compare human performance on the task of move prediction to the performance of MP*. Then in subsection 7.5.2 the move predictor is tested on professional games. Finally, in subsection 7.5.3 it is tested by playing against the program GNU Go.

7.5.1 Human performance with full-board information

We compared the performance of MP* with that of human performance. For this we selected three games played by strong players (3d*-5d* IGS). The three games were replayed by a number of human amateur Go players, all playing black. The task faced by the human was identical to that of the neural predictor, the main difference being that the human had access to complete full-board information. At each move the human player was instructed to choose between two moves: one of the two moves was the expert move, the other was a move randomly selected within a Manhattan distance of 3 from the expert move.

Table 7.4 shows the results achieved by the human players. The players are ordered according to their (Dutch) rating, shown in the first column. It should be noted that some of the ratings might be off by one or two grades, which is especially true for the low-ranked kyu players (5-15k). Only of the dan-level ratings we can be reasonably sure, since they are regulated by official tournament results. The next three columns contain the scores of the human players on the three games, and the last column contains their average scores over all moves. (Naturally all humans were given the exact same set of choices, and were not exposed to these games before.)

From Table 7.4 we estimate that dan-level performance lies somewhere around 94%. Clearly there is still a significant variation most likely related to some in-

rating	game 1 (%)	game 2 (%)	game 3 (%)	average (%)
3 dan	96.7	91.5	89.5	92.4
2 dan	95.8	95.0	97.0	95.9
2 kyu	95.0	91.5	92.5	92.9
MP*	90.0	89.4	89.5	89.6
2 kyu	87.5	90.8	n.a.	89.3
5 kyu	87.5	84.4	85.0	85.5
8 kyu	87.5	85.1	86.5	86.3
13 kyu	83.3	75.2	82.7	80.2
14 kyu	76.7	83.0	80.5	80.2
15 kyu	80.0	73.8	82.0	78.4

Table 7.4: Human and computer (MP*) performance on move prediction.

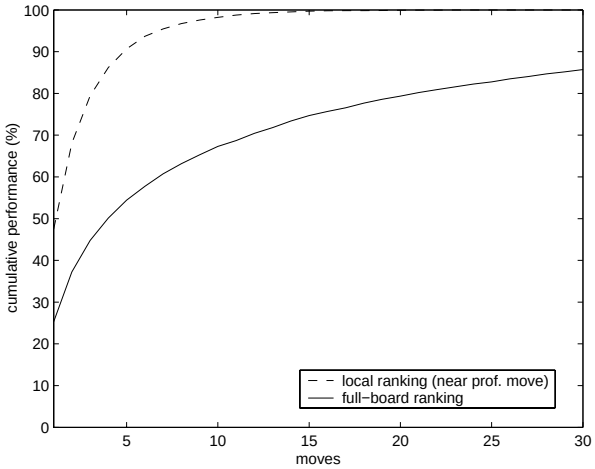


Figure 7.4: Ranking professional moves on 19×19.

herent freedom for choosing between moves that are (almost) equally good. Strong kyu level is somewhere around 90%, and as players get weaker we see performance dropping even below 80%. On the three games our move predictor (MP*) scored an average of 89.6% correct predictions thus placing it in the region of strong kyu-level players.

7.5.2 Testing on professional games

19×19 games

The performance of MP* was tested on 52 professional games played for the title matches of recent Kisei, Meijin, and Honinbo Tournaments. The Kisei, Meijin, and Honinbo are the most prestigious titles in Japan with total first-prize money of about US$ 600,000. The 52 games contained 11,460 positions with 248 legal moves on average (excluding the pass move). For each position MP* was used to rank all legal moves. We calculated the probability that the professional move was among the first n moves (cumulative performance). In Figure 7.4 the cumulative performance of the ranking is shown for the full board as well as for the local neighbourhoods (within a Manhattan distance of 3 from the professional move).

In local neighbourhoods the predictor ranked 48% of the professional moves first. On the full board the predictor ranked 25% of the professional moves first, 45% within the best three, and 80% in the top 20. The last 20% was in a long tailed distribution reaching 99% at about 100 moves.

In an experiment performed ten years ago by Müller and reported in [124] the program Explorer ranked one third of the moves played by professionals among its top three choices. Another third of the moves was ranked between 4 and 20, and the remaining third was either ranked below the top twenty or not

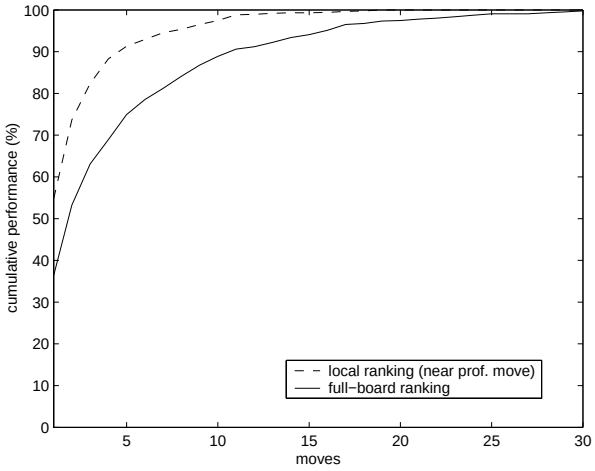


Figure 7.5: Ranking professional moves on 9×9.

considered at all. Though the comparison may be somewhat unfair, due to the fact that EXPLORER was not optimised for predicting professional moves, it still seems that significant progress has been made.

9 × 9 games

For fast games, and for teaching beginners, the game of Go is often played on the 9×9 board. This board has a reduced state space, a reduced branching factor, and a shorter game length, which result in a complexity between Chess and Othello [28]. Yet despite the reduced complexity the *current* 9×9 Go programs perform nearly as bad as Go programs on the 19×19 board.

We tested the performance of MP* (which was re-trained on amateur 9×9 games) on 17 professional 9×9 games. The games contained 862 positions with 56 legal moves on average (excluding the pass move). Figure 7.5 shows the cumulative performance of the ranking for the full board as well as for the local neighbourhoods (again within a Manhattan distance of 3 from the professional move). On the full 9×9 board the predictor ranked 37% of the professional moves first and over 99% of the professional moves in the top 25.

7.5.3 Testing by actual play

To assess the strength of our predictor in practice we tested it against GNU GO version 3.2. This was done by always playing the first-ranked move. Despite of many good moves in the opening and middle-game MP* lost all games. Thus, the move predictor in itself is not sufficient to play a strong game. The main handicap of the move predictor was that it did not understand (some of) the tactical fights. Occasionally this resulted in the loss of large groups, and poor play in the endgame. Another handicap was that always playing the first-ranked

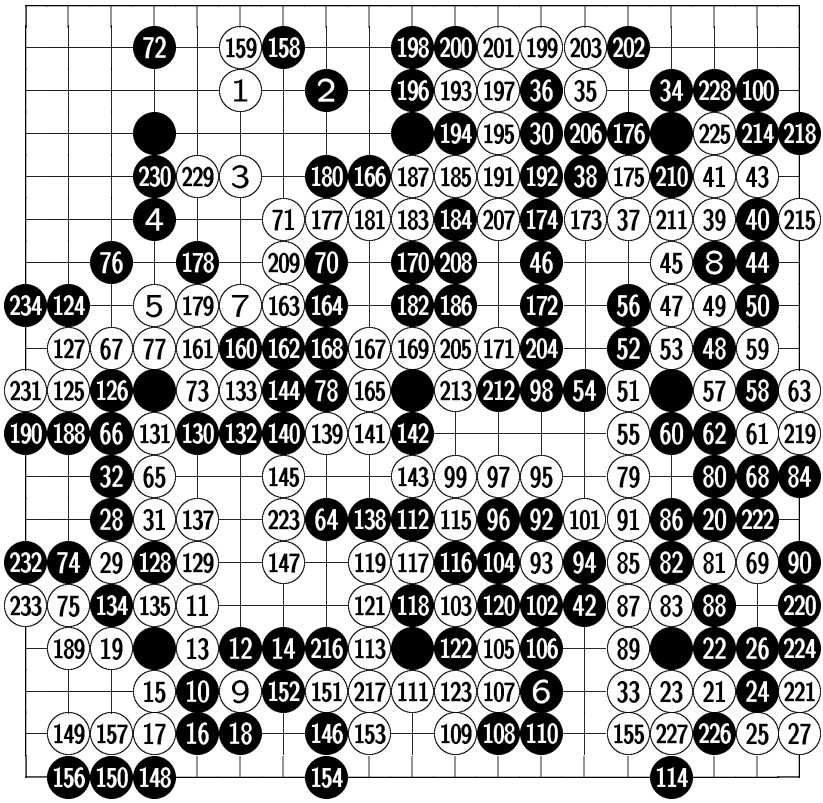


Figure 7.6: Nine-stone handicap game against GNU Go (white 136 at 29).

move often turned out to be too passive. (The program followed GNU Go's moves and seldom took initiative in other regions of the board.)

We hypothesised that a balanced combination of MP* with a decent tactical search procedure would have a significant impact on the performance, and in particular on the quality of the fights. However, since we did not have a reasonable search procedure (and evaluation function) available at the time, we did not explore this idea. As an alternative we tested some games where the author (at the time a strong kyu level player) selected moves from the first n candidate moves ranked by MP*. Playing with n equals ten we were able to defeat GNU Go even when it played with up to five handicap stones. With n equals twenty the strength of our combination increased even further. In Figure 7.6 a game is shown where GNU Go played Black with nine handicap stones against the author selecting from the first twenty moves. The game is shown up to move 234, where White is clearly ahead. After some small endgame fights White convincingly won the game with 57.5 points.

The results indicate that, at least against other Go programs, a relatively small set of high-ranked moves is sufficient to play a strong game.

7.6 Chapter conclusions

We have presented a system that learns to predict moves in the game of Go from observing strong human play. The performance of our best move predictor (MP*) is, at least in local regions, comparable to that of strong kyu-level players. Although the move predictor in itself is not sufficient to play a strong game, selecting from only a small number of moves as proposed by MP* is sufficient to defeat other Go programs, even at high handicaps.

The training algorithm presented here is more efficient than standard fixed-target implementations. This is mainly due to the avoidance of needless weight adaptation when rankings are correct. As an extra bonus, our training method reduces the number of gradient calculations as performance grows, thus speeding up the training. A major contribution to the performance is the use of feature-extraction methods. Feature extraction reduces the training time while increasing the quality of the predictor. Together with a sensible scaling of the original features and an optional second-phase training, superior performance over direct-training schemes can be obtained.

The predictor can be used for move ordering and forward pruning in a full-board search. The performance obtained on ranking professional moves indicates that a large fraction of the legal moves may be pruned directly without any significant risk. In particular, our results against GNU Go indicate that a relatively small set of high-ranked moves is sufficient to play a strong game.

On a 1 GHz PC our system evaluates moves with a speed in the order of roughly 5000 moves per second. This translates to around 0.05 seconds for a full-board ranking. As a consequence our approach may not be directly applicable for deep searches. The speed can however be increased greatly by parallel computation. Trade-offs between speed and predictive power are also possible since the number of hidden units and the dimensionality of the raw feature vector both scale linearly with computation time.

Regarding our second research question (see 1.3) we conclude that, for the task of move prediction, supervised learning techniques can provide a performance at least comparable to strong kyu-level players. The performance was obtained with a representation consisting of a relatively simple set of features, thus ignoring a significant amount of information which can be obtained by more extensive (full-board) analysis or by specific goal-directed searches. Consequently, there is still significant room for improving the performance, possibly even into the strong dan-level region.

Future research

Experiments showed that MP* performs well on the prediction of moves which are played in strong human games. The downside however is that MP* cannot be trusted (yet) in odd positions which do not show up in (strong) human games. Future work should therefore focus on ensuring reliability of the move predictor in more odd regions of the Go state space.

It may be interesting to train the move predictor further through some type

of Q-learning. In principle Q-learning works regardless of who is selecting the moves. Consequently, training should work both by self-play (in odd positions) and by replaying human games. Furthermore, since Q-learning does not rely on the assumption of the optimality of human moves, it may be able to solve possible inconsistencies in its current knowledge (due to the fact that some human moves were bad).

Finally, future research should focus on the application of our move predictor for move ordering and forward pruning in full-board search. Preliminary results suggested that it can greatly improve the search in particular if it can be combined with a sensible full-board evaluation function.

Acknowledgements

We are grateful to all Go players that helped us perform the experiments reported in subsection 7.5.1.

Chapter 8

Scoring final positions

This chapter is based on E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to score final positions in the game of Go. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 143–158. Kluwer Academic Publishers, Boston, MA, 2003. An extended version is to appear in [191].[1]

The best computer Go programs are still in their infancy. They are no match even for human amateurs of only moderate skill. Partially this is due to the complexity of Go, which makes brute-force search techniques infeasible on the 19×19 board. As stated in subsection 7.5.2, on the 9×9 board, which has a complexity between Chess and Othello [28], the *current* Go programs perform nearly as bad. The main reason lies in the lack of good positional evaluation functions. Many (if not all) of the current top programs rely on (huge) static knowledge bases derived from the programmers' Go skills and Go knowledge. As a consequence the top programs are extremely complex and difficult to improve. In principle a learning system should be able to overcome this problem.

In the past decade several researchers have used machine-learning techniques in Go. After Tesauro's [170] success story many researchers, including Dahl [51], Enzenberger [71] and Schraudolph et al. [159], have applied Temporal Difference (TD) learning for learning evaluation functions. Although TD-learning is a promising technique, which was underlined by NeuroGo's silver medal in the 9×9 Go tournament at the 8^{th} Computer Olympiad in Graz [181], there has not been a major breakthrough, such as in Backgammon, and we believe that this will remain unlikely to happen in the near future as long as most learning is done from self-play or against weak opponents.

Over centuries humans have acquired extensive knowledge of Go. Since the knowledge is implicitly available in the games of human experts, it should be possible to apply machine-learning techniques to extract that knowledge from game records. So far, game records were only used successfully for move

[1] The author would like to thank Kluwer Academic Publishers, Elsevier Science and his co-authors for permission to reuse relevant parts of the article in this thesis.

prediction [51, 70, 182]. However, we are convinced that much more can be learned from these game records.

One of the best sources of game records on the Internet is the No Name Go Server game archive [136]. NNGS is a free on-line Go club where people from all over the world can meet and play Go. All games played on NNGS since 1995 are available on-line. Although NNGS game records contain a wealth of information, the automated extraction of knowledge from these games is a non-trivial task at least for the following three reasons.

Missing Information. Life-and-death status of blocks is not available. In scored games only a single numeric value representing the difference in points is available.

Unfinished Games. Not all games are scored. Human games often end by one side resigning or abandoning the game without finishing it, which often leaves the status of large parts of the board unclear. [2]

Bad Moves. During the game mistakes are made which are hard to detect. Since mistakes break the chain of optimal moves it can be misleading (and incorrect from a game-theoretical point of view) to relate positions before the mistake to the final outcome of the game.

The first step towards making the knowledge in the game records accessible is to obtain reliable scores at the end of the game. Reliable scores require correct classifications of life and death. This chapter focuses on determining life and death for final positions. By focusing on final positions we avoid the problem of unfinished games and bad moves during the game, which will be addressed in the next chapter.

It has been pointed out by Müller [125] that proving the score of final positions is a hard task. For a set of typical human final positions, Müller showed that extending Benson's techniques for proving life and death [14] with a more sophisticated static analysis and search, still leaves around 75% of the board points unproven. Heuristic classification of his program EXPLORER classified most blocks correctly, but still left some regions unsettled (and to be played out further). Although this may be appropriate for computer-computer games, it can be annoying in human-computer games, especially under the Japanese rules which penalise playing more stones than necessary.

Since proving the score of most final positions is not (yet) an option, we focus on learning a heuristic classification. We believe that a learning algorithm for scoring final positions is important because: (1) it provides a more flexible framework than the traditional hand-coded static knowledge bases, and (2) it is a necessary first step towards learning to evaluate non-final positions. In general such an algorithm is good to have because: (1) large numbers of game records are hard to score manually, (2) publicly available programs still make too many

[2] In professional games that are not played on-line similar problems can occur when the final reinforcing moves are omitted because they are considered obvious.

mistakes when scoring final positions, and (3) it can avoid unnecessarily long human-computer games.

The remainder of this chapter is organised as follows. Section 8.1 discusses the scoring method. Section 8.2 presents the learning task. Section 8.3 introduces the representation. Section 8.4 provides details about the data set. Section 8.5 reports our experiments. Finally, section 8.6 presents our conclusions and on-going work.

8.1 The scoring method

In this thesis we use area scoring, introduced in subsection 2.2.4. The process of applying area scoring to a final position works as follows. First, the life-and-death status of blocks of connected stones is determined. Second, dead stones are removed from the board. Third, each empty point is marked Black, White, or neutral. The non-empty points are already marked by their colour. The empty points can be marked by flood filling or by distance. Flood filling recursively marks empty points to their adjacent colour. In the case that a flood fill for Black overlaps with a flood fill for White the overlapping region becomes neutral. (As a consequence all non-neutral empty regions must be completely enclosed by one colour.) Scoring by distance marks each point based on the distance towards the nearest remaining black or white stone(s). If the point is closer to a black stone it is marked black, if the point is closer to a white stone it is marked white, otherwise (if the distance is equal) the point does not affect the score and is marked neutral. Finally, the difference between black and white points, together with a possible komi, determines the outcome of the game.

In final positions scoring by flood filling and scoring by distance should give the same result. If the result is not the same, there are large open regions with unsettled interior points, which usually means that some stones should have been removed or some points could still be gained by playing further. Comparing flood filling with scoring by distance, to detect large open regions, is a useful check to find out whether the game is finished and scored correctly.

8.2 The learning task

The task of learning to score comes down to learning to determine which blocks of connected stones are dead and should be removed from the board. This can be learned from a set of labelled final positions, for which the labels contain the colour controlling each point. A straightforward implementation would be to learn classifying all blocks based on the labelled points. However, for some blocks this is not a good idea because their status can be irrelevant and forcing them to be classified just complicates the learning task.

8.2.1 Which blocks to classify?

For arriving at a correct score we require correct classifications for only two types of blocks. The first type is dead in the opponent's area. The second type is alive and at the border of friendly area. (Notice that, for training, the knowledge where the border is will be obtained from labelled game records.) The distinction between block types is illustrated in Figure 8.1. Here all marked stones must be classified. The stones marked by triangles must be classified alive. The stones marked by squares must be classified dead. The unmarked stones are irrelevant for scoring because they are not at the border of their area and their capturability does not affect the score. For example, the two black stones in the top-left corner kill the white block and are in Black's area. However, White can always captured them, so forcing them to be classified as alive or dead is misleading and even unnecessary. (The stones in the bottom left corner are alive in seki because neither side can capture. The two white stones in the upper right corner are adjacent to two neutral points (dame) and therefore also at the border of White's region.)

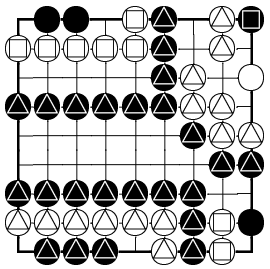


Figure 8.1: Blocks to classify.

8.2.2 Recursion

Usually blocks of stones are not alive on their own. Instead they form chains or groups which are only alive in combination with other blocks. Their status also may depend on the status of neighbouring blocks of the opponent, i.e., blocks can live by capturing the opponent. (Although one might be tempted to conclude that life and death should be dealt with at the level of groups this does not really help because the human notion of a group is not well defined, difficult to program, and may even require an underlying notion of life and death.)

Because life and death of blocks is strongly related to the life and death of other blocks the status of other (usually nearby) blocks has to be taken into account. Partially this can be done by including features for nearby blocks in the representation. In addition, it seems natural to consider a recursive framework for classification which employs the predictions for other blocks to improve performance iteratively. In our implementation this is done by training a cascade of classifiers which use previous predictions for other blocks as additional input features.

8.3 Representation

In this section we will present the representation of blocks for classification. Several representations are possible and used in the field. The most primitive representations typically employ the raw board directly. A straightforward implementation is to concatenate three bitboards into a feature vector, for which

the first bitboard contains the block to be classified, the second bitboard contains other friendly blocks, and the third bitboard contains the enemy blocks. Although this representation is complete, in the sense that all relevant information is preserved it is unlikely to be efficient because of the high dimensionality and lack of topological structure.

8.3.1 Features for Block Classification

A more efficient representation employs a set of features based on simple measurable geometric properties, some elementary Go knowledge and some handcrafted specialised features. Several of these features are typically used in Go programs to evaluate positions [45, 73]. The features are calculated for: (1) single friendly blocks, (2) single opponent blocks, (3) multiple blocks in chains, and (4) colour-enclosed regions (CERs).

General features

For each block our representation consists of the following features (all features are single scalar values unless stated otherwise).

- *Size* measured in occupied points.
- *Perimeter* measured in number of adjacent points.
- *Opponents* are the occupied adjacent points.
- (*First-order*) *liberties* are the free (empty) adjacent points.
- *Protected liberties* are the liberties which normally should not be played by the opponent, because of suicide or being directly capturable.
- *Auto-atari liberties* are liberties which by playing them reduce the liberties of the block from 2 to 1; it means that the block would become directly capturable (such liberties are protected for an adjacent opponent block).
- *Second-order liberties* are the empty points adjacent to but not part of the liberties.
- *Third-order liberties* are the empty points adjacent to but not part of the first-order and second-order liberties.
- *Number of adjacent opponent blocks*
- *Local majority* is the number of friendly stones minus the number of opponent stones within a Manhattan distance of 2 from the block.
- *Centre of mass* represented by the average distance of stones in the block to the closest and second-closest edge (using floating-point scalars).
- *Bounding box size* is the number of points in the smallest rectangular box that can contain the block.

Colour-enclosed regions

Adjacent to each block are colour-enclosed regions. CERs consist of connected empty and occupied points, surrounded by stones of one colour. (Notice that regions along the edge, such as an eye in the corner, are also enclosed). It is important to know whether an adjacent CER is fully accessible, because a fully accessible CER surrounded by safe blocks provides at least one sure liberty (the surrounding blocks are safe when they all have at least two sure liberties). To detect fully accessible regions we use so-called miai strategies as applied by Müller [125]. In addition to Müller's original implementation we (1) add miai-accessible interior empty points to the set of accessible liberties, and (2) use protected liberties for the chaining. An example of a fully accessible CER is shown in Figure 8.2. Here the idea is that if White plays on a marked empty point, Black replies on the other empty point marked by the same letter. By following this miai strategy Black is guaranteed to be able to occupy or become adjacent to all points in the region, i.e., all empty points in Figure 8.2 that are not directly adjacent to black stones are miai-accessible interior empty points; the points on the edge marked 'b' and 'e' were not used in Müller's original implementation [128]. Often it is not possible to find a miai strategy for the full region, in which case we call the CER partially accessible. In Figure 8.3 an example of a partially accessible CER is shown. In this case the 3 points marked 'x' form the inaccessible interior for the given miai strategy.

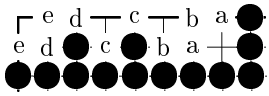


Figure 8.2: Fully accessible CER.

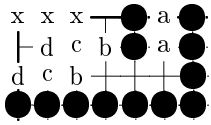


Figure 8.3: Partially accessible CER.

Analysis of the CERs can provide us with several interesting features. However, the number of regions is not fixed, and our representation requires a fixed number of features. Therefore we decided to sum the features over all regions. For fully accessible CERs we include the following features.

- *Number of regions*
- *Size* [3]
- *Perimeter*
- *Number of split points* in the CER. Split points are crucial points for preserving connectedness in the local 3×3 window around the point. (The region could still be connected by a big loop outside the local 3×3 window.) Examples are shown in Figure 8.4.

[3] Although regions may contain stones we deal with them as blocks of connected intersections regardless of the colour. Calculations of the various features, such as size, perimeter, and split points, are performed analogously to the calculations for normal blocks of one colour.

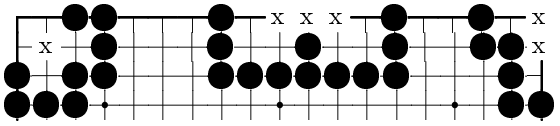


Figure 8.4: Split points marked with x.

For partially accessible CERs we include the following features.

- *Number of partially accessible regions*
- *Accessible size*
- *Accessible perimeter*
- *Size* of the inaccessible interior.
- *Perimeter* of the inaccessible interior.
- *Split points* of the inaccessible interior.

Eyespace

Another way to analyse CERs is to look for possible eyespace. Points forming the eyespace should be empty or contain capturable opponent stones. Empty points directly adjacent to opponent stones are not part of the eyespace. Points on the edge with one or more diagonally adjacent alive opponent stones and points with two or more diagonally adjacent alive opponent stones are false eyes. False eyes are not part of the eyespace (we ignore the unlikely case where a big loop upgrades false eyes to true eyes). For example, in Figure 8.5 the points marked 'e' belong to Black's eyespace and the points marked 'f' are false eyes for White. Initially we assume all diagonally adjacent opponent stones to be alive. However, in the recursive framework (see below) the eyespace is updated based on the status of the diagonally adjacent opponent stones after each iteration.

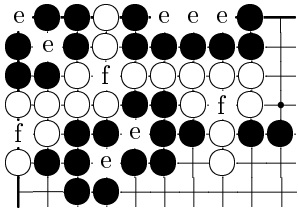


Figure 8.5: True and false eyespace.

For directly adjacent eyespace of the block we include two features.

- *Size*
- *Perimeter*

Optimistic chain

Since we are dealing with final positions it is often possible to use the optimistic assumption that all blocks with shared liberties can form a chain (during the game this assumption can be dangerous because the chain may be split). Examples of a black and a white optimistic chain are shown in Figure 8.6. For the block's optimistic chain we include the following features.

- *Number of blocks*
- *Size*
- *Perimeter*
- *Split points*
- *Number of adjacent CERs*
- *Number of adjacent CERs with eyespace*
- *Number of adjacent CERs, fully accessible* from at least one block.
- *Size of adjacent eyespace*
- *Perimeter of adjacent eyespace* (Again, in the case of multiple connected regions for the eyespace, size and perimeter are summed over all regions.)
- *External opponent liberties* are liberties of adjacent opponent blocks that are not accessible from the optimistic chain.

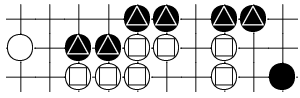


Figure 8.6: Marked optimistic chains.

Weak opponent blocks

Adjacent to the block in question there may be opponent blocks. For the weakest (measured by the number of liberties) directly adjacent opponent block we include the following features.

- *Perimeter*
- *Liberties*
- *Shared liberties*
- *Split points*
- *Perimeter of adjacent eyespace*

The same features are also included for the second-weakest directly adjacent opponent block and the weakest opponent block directly adjacent to or sharing liberties with the optimistic chain of the block in question (so the weakest directly adjacent opponent block may be included twice).

Disputed territory

By comparing a flood fill starting from Black with a flood fill starting from White we find unsettled empty regions which are disputed territory (assuming all blocks are alive). If the block is adjacent to disputed territory we include the following features.

- *Direct liberties* in disputed territory.
- *Liberties of all friendly blocks* in disputed territory.
- *Liberties of all enemy blocks* in disputed territory.

8.3.2 Additional features for recursive classification

For the recursive classification we use the predicted values of previous classifications, which are floating-point scalars in the range between 0 (dead) and 1 (alive), to construct the following six additional features.

- *Predicted value* of the strongest friendly block with a shared liberty.
- *Predicted value* of the weakest adjacent opponent block.
- *Predicted value* of the second-weakest adjacent opponent block.
- *Average predicted value* of the weakest opponent block's optimistic chain.
- *Adjacent eyespace size* of the weakest opponent block's optimistic chain.
- *Adjacent eyespace perimeter* of the weakest opponent block's optimistic chain.

Next to these additional features the predictions are also used to update the eyespace, i.e., dead blocks can become eyespace for the side that captures, alive blocks cannot provide eyespace, and diagonally adjacent dead opponent stones are not counted for detecting false eyes.

8.4 The data set

In the experiments we used game records obtained from the NNGS archive [136]. All games were played on the 9×9 board between 1995 and 2002. We only considered games that were played to the end and scored, thus ignoring unfinished or resigned games. Since the game records only contain a single numeric value for the score, we had to find a way to label all blocks.

8.4.1 Scoring the data set

For scoring the data set we initially used a combination of GNU Go (version 3.2) [74] and manual labelling. Although GNU Go has the option to finish games and label blocks the program could not be used without human supervision. The reasons for this are threefold: (1) bugs, (2) the inherent complexity of the task, and (3) the mistakes made by weak human players who ended the game in positions that were not final, or scored them incorrectly. Fortunately, nearly all mistakes were easily detected by comparing GNU Go's scores and the labelled boards with the numeric scores stored in the game records.[4] As an additional check all boards containing open regions with unsettled interior points (where flood filling does not give the same result as distance-based scoring) were also inspected manually.

Since the scores did not match in many positions the labelling proved to be very time consuming. We therefore only used GNU Go to label the games played in 2002 and 1995. With the 2002 games a classifier was trained. When we tested the performance on the 1995 games it outperformed GNU Go's labelling. Therefore our classifier replaced GNU Go for labelling the other games (1996-2001), retraining it each time a new year was labelled. Although this sped up the process it still required a fair amount of human intervention mainly because of games that contained incorrect scores in their game record. A few hundred games had to be thrown out completely because they were not finished, contained illegal moves, contained no moves at all (for at least one side), or both sides were played by the same player. In a small number of cases, where the last moves would have been trivial but not actually played, we made the last few moves manually.

Eventually we ended up with a data set containing 18,222 final positions. Around 10% of these games were scored incorrectly (by the players) and were inspected manually. (Actually the number of games we inspected is significantly higher because of the games that were thrown out and because both our initial classifiers and GNU Go made mistakes.) On average the final positions contained 5.8 alive blocks, 1.9 dead blocks, and 2.7 irrelevant blocks. (In the case that one player gets the full board we count all blocks of this player as irrelevant, because there is no border. Of course, in practice at least one block should be classified as alive, which appears to be learned automatically without any special attention.)

Since the Go scores on the 9×9 board range from -81 to $+81$ the chances of an incorrect labelling leading to a correct score are low, nevertheless it could not be ruled out completely. On inspecting an additional 1% of the positions randomly we found none that were labelled incorrectly. Finally, when all games were labelled, we re-inspected all positions for which our best classifier seemed to predict an incorrect score. This final pass detected 42 positions (0.2%) that were labelled incorrectly, mostly because our initial classifiers had made the same mistakes as the players who scored the games.

[4] All differences caused by territory scoring were filtered out automatically, except when dealing with eyes in seki.

8.4.2 Statistics

Since many game records contained incorrect scores we looked for reasons and gathered statistics. The first thing that came to mind is that weak players might not know how to score. Therefore in Figure 8.7 the percentage of incorrectly scored games related to the strength of the players is shown. (Although in each game only one side may have been responsible for the incorrect score, we always assigned blame to both sides.) The two marker types distinguish between rated and unrated players. Although unrated players have a value for their rating, it is an indication given by the player and not by the server. Only after playing sufficiently many games the server assigns players a rating.

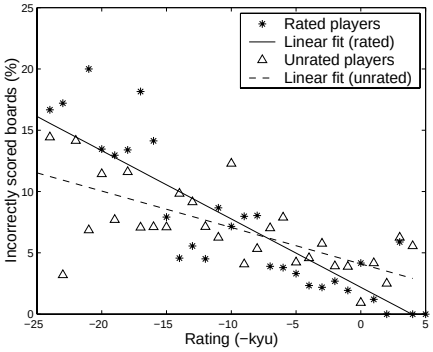


Figure 8.7: Incorrect scores.

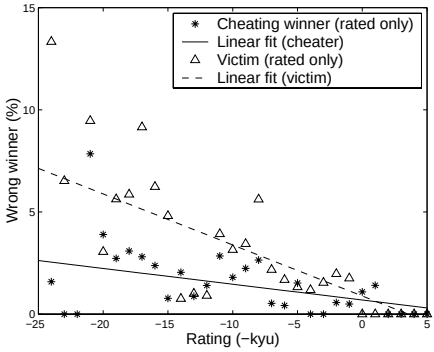


Figure 8.8: Incorrect winners.

Although a significant number of games are scored incorrectly this is usually not considered a problem when the winner is correct. (Players typically forget to remove some stones when they are far ahead.) Figure 8.8 shows how often incorrect scoring by rated players converts a loss into a win (cheater) or a win into a loss (victim).

It should be noted that the percentages in Figures 8.7 and 8.8 were weighted over all games, regardless of who was the player. Therefore, they do not necessarily reflect the probabilities for individual players, i.e., the statistics can be dominated by a small group of players that played many games. This group at least contains some computer players, which have a tendency to get robbed of their points in the scoring phase. Hence, we calculated some statistics that were normalised over individual players, e.g., statistics of players who played hundreds of games were weighted equal to the statistics of players who played only a small number of games. Thereupon we found that for rated players the average probability of scoring a game incorrectly is 4.2%, the probability of cheating (the incorrect score converts a loss into a win) is 0.66%, and the probability of getting cheated is 0.55%. For unrated players the average probability of scoring a game incorrectly is 11.2%, the probability of cheating is 2.1%, and the probability of getting cheated is 1.1%. The fact that, when we normalise over players, the probability of getting cheated is lower than the probability of

cheating is the result of a small group of players (several of them are computer programs) who systematically lose points in the scoring phase, and a larger group of players who take advantage of that fact.

8.5 Experiments

In this section experimental results are presented for: (1) selecting a classifier, (2) performance of the representation, (3) recursive performance, (4) full-board performance, and (5) performance on the 19×19 board. Unless stated otherwise the various training and validation sets, used in the experiments, were extracted from games played between 1996 and 2002. The test set was always the same, containing 7149 labelled blocks extracted from 919 games played in 1995.

8.5.1 Selecting a classifier

An important choice is selecting a good classifier. In pattern recognition there is a wide range of classifiers to choose from [93]. We tested a number of well-known classifiers, introduced in section 6.2, for their performance (without recursion) on data sets of 100, 1000, and 10,000 examples. The classifiers are: nearest mean classifier (NMC), linear discriminant classifier (LDC), logistic linear classifier (LOGLC), quadratic discriminant classifier (QDC), nearest neighbour classifier (NNC), k-nearest neighbours classifier (KNNC), backpropagation neural net classifier with momentum and adaptive learning (BPNC), Levenberg-Marquardt neural net classifier (LMNC), and resilient propagation neural net classifier (RPNC). Some preliminary experiments with a support vector classifier, decision tree classifiers, a Parzen classifier, and a radial basis neural net classifier were not pursued further because of excessive training times and/or poor performance. All classifiers except the neural net classifiers, for which we directly used the standard Matlab toolbox, were used as implemented in PRTools3 [65].

The results, shown in Table 8.1, indicate that the performance first of all depends on the size of the training set. The linear classifiers perform better than the quadratic classifier and nearest neighbour classifiers. For large data sets training KNNC is very slow because it takes a long time to find an optimal value of the parameter k. The number of classifications per second of (K)NNC is also low because of the large number of distances that must be computed (all training examples are stored). Although editing and condensing the data set still might improve the performance of the nearest neighbour classifiers, we did not investigate them further.

The best classifiers are the neural network classifiers. It should however be noted that their performance may be slightly overestimated with respect to the size of the training set, because we used an additional validation set to stop training (this was not possible for the other classifiers because they are not trained incrementally). The logistic linear classifier performs nearly as well as

Classifier	Training size	Training error (%)	Test error (%)	Training time (s)	Classi. speed (s^{-1})
NMC	100	2.8	3.9	0.0	4.9×10^4
	1000	4.0	3.8	0.1	5.2×10^4
	10,000	3.8	3.6	0.5	5.3×10^4
LDC	100	0.7	3.0	0.0	5.1×10^4
	1000	2.1	2.0	0.1	5.2×10^4
	10,000	2.2	1.9	0.9	5.3×10^4
LOGLC	100	0.0	9.3	0.2	5.2×10^4
	1000	0.0	2.6	1.1	5.2×10^4
	10,000	1.0	1.2	5.6	5.1×10^4
QDC	100	0.0	13.7	0.1	3.1×10^4
	1000	1.0	2.1	0.1	3.2×10^4
	10,000	1.9	2.1	1.1	3.2×10^4
NNC	100	0.0	18.8	0.0	4.7×10^3
	1000	0.0	13.5	4.1	2.4×10^2
	10,000	0.0	10.2	4.1×10^3	2.4×10^0
KNNC	100	7.2	13.1	0.0	4.8×10^3
	1000	4.2	4.4	1.0×10^1	2.4×10^2
	10,000	2.8	2.8	9.4×10^3	2.6×10^0
BPNC	100	0.5	3.6	2.9	1.8×10^4
	1000	0.2	1.5	1.9×10^1	1.8×10^4
	10,000	0.5	1.0	1.9×10^2	1.9×10^4
LMNC	100	2.2	7.6	2.6×10^1	1.8×10^4
	1000	0.7	2.8	3.2×10^2	1.8×10^4
	10,000	0.5	1.2	2.4×10^3	1.9×10^4
RPNC	100	1.5	4.1	1.4	1.8×10^4
	1000	0.2	1.7	7.1	1.8×10^4
	10,000	0.4	1.1	7.1×10^1	1.9×10^4

Table 8.1: Performance of classifiers without recursion.

the neural network classifiers, which is quite an achievement considering that it is just a linear classifier.

The results of Table 8.1 were obtained with networks that employed one hidden layer containing 15 neurons with hyperbolic tangent sigmoid transfer functions. Since our choice for 15 neurons was quite arbitrary a second experiment was performed in which we varied the number of neurons in the hidden layer. In Figure 8.9 results are shown for the RPNC. The classification errors marked with triangles represent results for training on 5000 examples, the stars indicate results for training on 15,000 examples. The solid lines are measured on the independent test set, whereas the dash-dotted lines are obtained on the training set. The results show that even moderately sized networks easily overfit the data. Although the performance initially improves with the size of the network, it seems to level off for networks with over 50 hidden neurons (the stan-

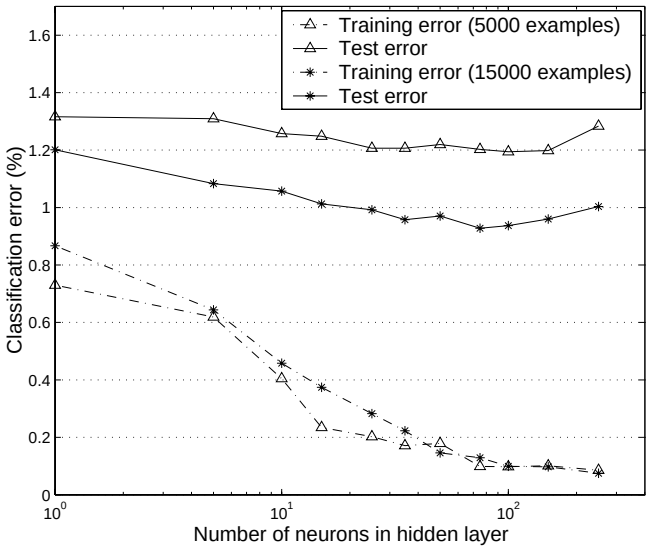


Figure 8.9: Sizing the neural network for the RPNC.

dard deviation is around 0.1%). Again, the key factor in improving performance clearly is in increasing the size of the training set.

8.5.2 Performance of the representation

In section 8.3 we claimed that a raw board representation is inefficient for predicting life and death. To validate this claim we measured the performance of such a representation and compared it to our specialised representation.

The raw representation consists of three concatenated bitboards, for which the first bitboard contains the block to be classified, the second bitboard contains other friendly blocks, and the third bitboard contains the enemy blocks. To remove symmetry the bitboards are rotated such that the centre of mass of the block to be classified is always in a single canonical region.

Since high-dimensional feature spaces tend to raise several problems which are not directly caused by the quality of the individual features we also tested two compressed representations. These compressed representations were generated by performing principal component analysis (PCA) (see 7.3.1) on the raw representation. For the first PCA mapping the number of features was chosen identical to our specialised representation. For the second PCA mapping the number of features was set to preserve 90% of the total variance.

The results, shown in Table 8.2, are obtained for the RPNC with 15, 35, and 75 neurons in the hidden layer, for training sets with 100, 1000, and 10,000 examples. All values are averages over 11 runs with different training sets, validation sets (same size as the training set), and random initialisations. The errors, measured on the test set, indicate that a raw representation alone re-

Training Size	Extractor	Test error 15 neurons (%)	Test error 35 neurons (%)	Test error 75 neurons (%)
100	-	29.1	26.0	27.3
100	pca1	22.9	22.9	22.3
100	pca2	23.3	24.3	21.9
1000	-	13.7	13.5	13.4
1000	pca1	16.7	16.2	15.6
1000	pca2	14.2	14.5	14.4
10,000	-	7.5	6.8	6.5
10,000	pca1	9.9	9.3	9.1
10,000	pca2	8.9	8.2	7.7

Table 8.2: Performance of the raw representation.

quires too many training examples to be useful in practice. Even with 10,000 training examples the raw representation performs much more weakly than our specialised representation with only 100 training examples. Simple feature-extraction methods such as principal component analysis do not seem to improve performance, indicating that preserved variance of the raw representation is relatively insignificant for determining life and death. (Some preliminary results for other feature-extraction methods used in the previous chapter were not encouraging either.)

8.5.3 Recursive performance

Our recursive framework for classification is implemented as a cascade of classifiers which use extra features, based on previous predictions as discussed in subsection 8.3.2, as additional input. The performance measured on an independent test set for the first 4 steps is shown for various sizes of the training set in Table 8.3. The results are averages of 5 runs with randomly initialised networks containing 50 neurons in the hidden layer (the standard deviation is around 0.1%).

The results show that recursive predictions improve the performance. However, the only significant improvement comes from the first iteration. The im-

Training Size	Direct error (%)	2-step error (%)	3-step error (%)	4-step error (%)
1000	1.93	1.60	1.52	1.48
10,000	1.09	0.76	0.74	0.72
100,000	0.68	0.43	0.38	0.37

Table 8.3: Recursive performance.

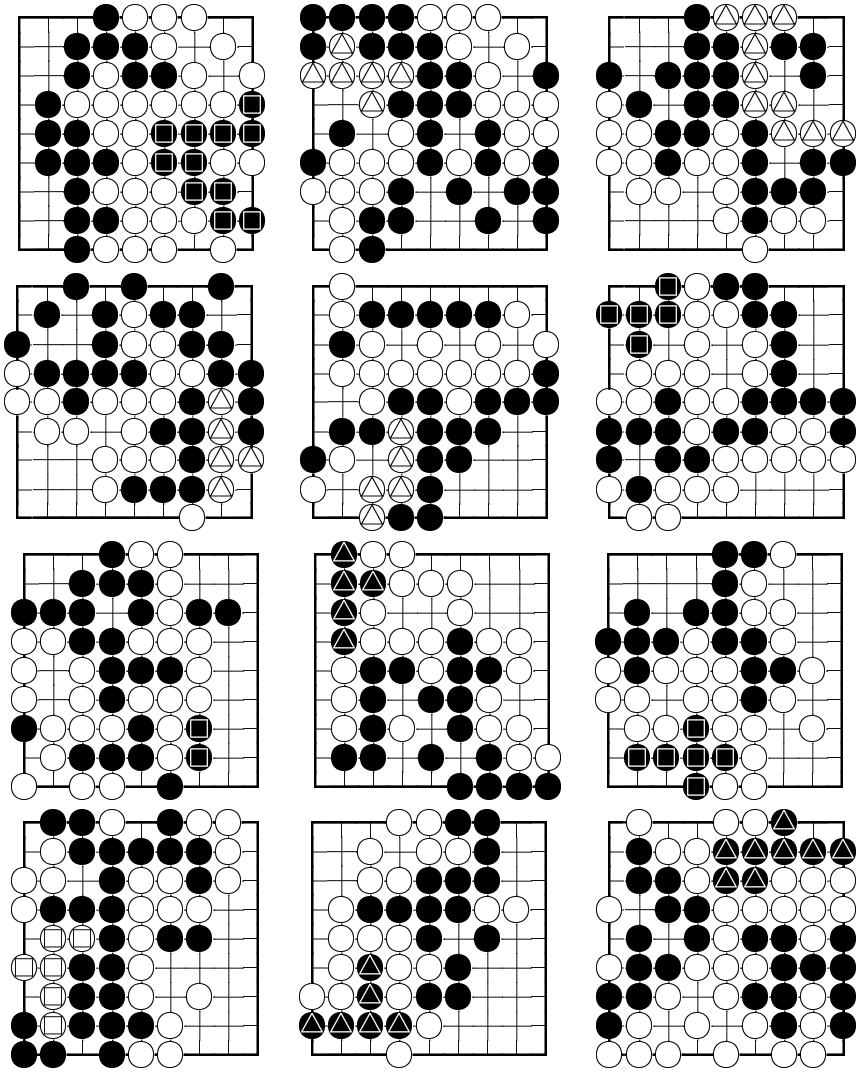


Figure 8.10: Examples of mistakes that are corrected by recursion.

provements are far from significant for the average 3-step and 4-step errors. The reason for this is that sometimes the performance got stuck or even worsened after the first iteration. Preliminary experiments suggest that large networks were more likely to get stuck after the first iteration than small networks, which might indicate some kind of overfitting. A possible solution to overcome this problem is to retrain the networks a number of times, and pick the best one based on the performance on the validation set. If we do this, our best networks trained on 100,000 training examples achieve a 4-step error of 0.25%. We refer

to the combination of the four cascaded classifier networks and the marking of empty intersections based on the distance to the nearest living block (which may be verified by comparing to flood filling, see section 8.1) by CSA* (Cascaded Scoring Architecture).

In Figure 8.10 we show twelve examples of mistakes that are made by direct classification without recursion, which can be corrected by using the 4-step recursion of CSA*. All marked blocks were initially classified incorrectly. Initially, the blocks marked with squares were classified as alive, and the blocks marked with triangles were classified as dead. After recursion this was corrected so that the blocks marked with squares are classified as dead, and the blocks marked with triangles are classified as alive.

8.5.4 Full-board performance

So far we have concentrated on the percentage of blocks that are classified correctly. Although this is an important measure it does not directly indicate how often boards will be scored correctly (a board may contain multiple incorrectly classified blocks). Further, we do not yet know what the effect is on the score in number of board points. Therefore we tested our classifiers on the full-board test positions, which were not used for training or validation.

For CSA* we found that 1.1% of the boards were scored incorrectly. For 0.5% of the boards the winner was not identified correctly. The average number of incorrectly scored board points (using distance-based scoring) was 0.15. However, in case a board is scored incorrectly it usually affects around 14 board points (which counts double in the numeric score).

In Figure 8.11 we show examples of the (rare) mistakes that are still made by the 4-step classification of CSA*. All marked blocks were classified incorrectly. The blocks marked with squares were incorrectly classified as alive. The blocks marked with triangles were incorrectly classified as dead. The difficult positions typically include seki, long chains connected by false eyes, bent four and similar looking shapes, and rare shapes such as ten-thousand year ko. In general we believe that many of these mistakes can be corrected by adding more training examples. However, for some positions it might be best to add new features or use a local search.

8.5.5 Performance on the 19 × 19 board

The experiments presented above were all performed on the 9×9 board which, as was pointed out before, is a challenging environment. Nevertheless, it is interesting to test whether (and if so to what extent) the techniques scale up to the 19×19 board. So far we did not focus on labelling large quantities of 19×19 games. Therefore, training directly on the 19×19 board was not an option. Despite of this we tested CSA*, which was trained using blocks observed on the 9×9 board, on the problem set *IGS_31_counted* from the Computer Go Test Collection. This set contains 31 labelled 19×19 games played by amateur dan players, and was used by Müller [125]. On the 31 final positions our 4-step

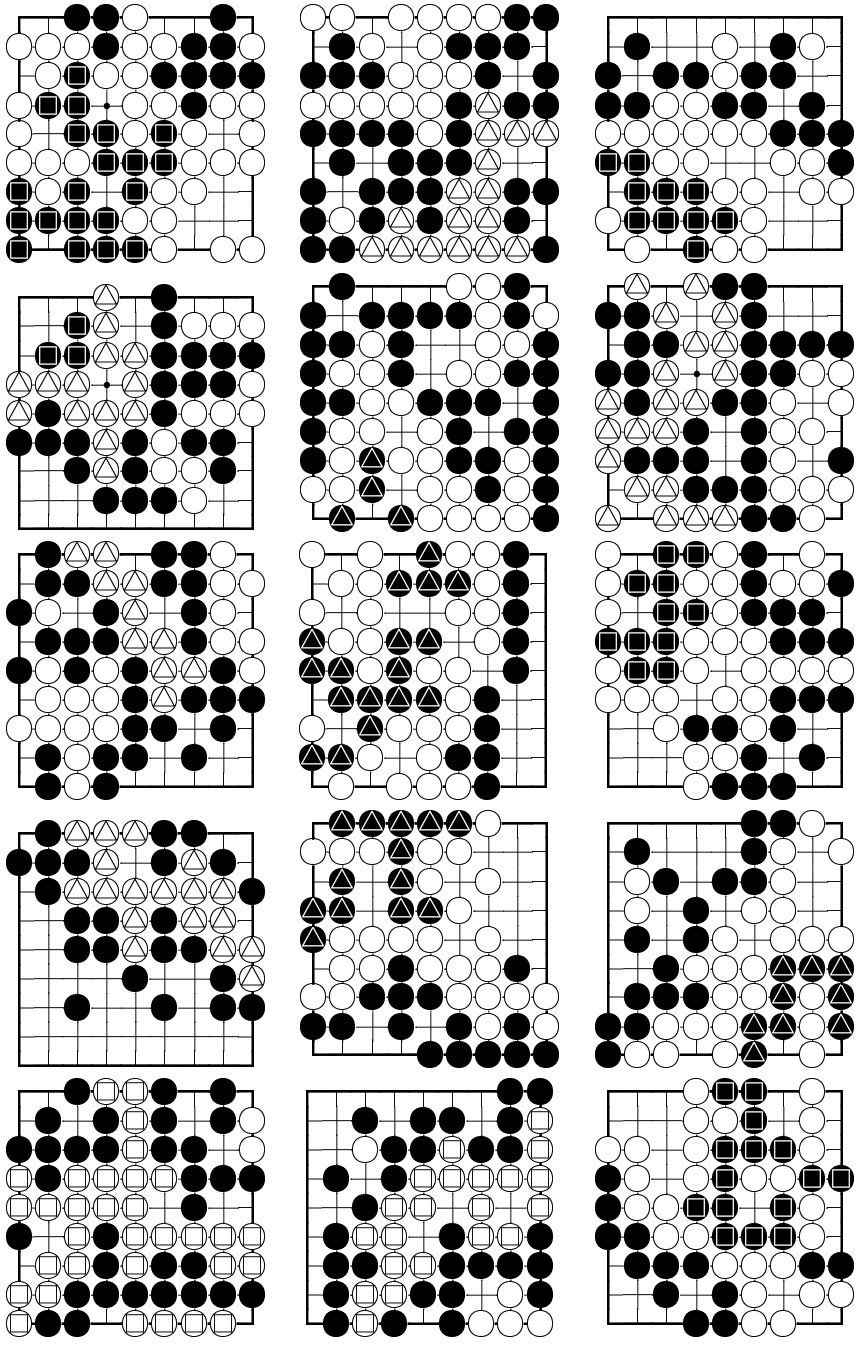


Figure 8.11: Examples of incorrectly scored positions.

classifier classified 5 blocks incorrectly (0.5% of all relevant blocks), and as a consequence 2 final positions were scored incorrectly. The average number of incorrectly scored board points was 2.1 (0.6%).

In his paper Müller [125] stated that the heuristic classification by his program EXPLORER classified most blocks correctly. Although we do not know the exact performance of EXPLORER we believe it is safe to say that CSA*, which classified 99.5% of all blocks correctly, is performing at least at a comparable level. Furthermore, since our system was not trained explicitly for 19×19 games there may still be significant room for improvement.

8.6 Chapter conclusions

We have developed a Cascaded Scoring Architecture (CSA*) that learns to score final positions from labelled examples. On unseen game records CSA* scored around 98.9% of the positions correctly without any human intervention. Compared to the average rated player on NNGS, who has a rating of 7 kyu for scored 9×9 games, we may conclude that CSA* is more accurate at removing all dead blocks, and performs comparably on determining the correct winner.

Regarding our second research question (see 1.3), and the questions posed in section 6.3, we conclude that for the task of scoring final positions supervised learning techniques can provide a performance at least comparable to reasonably strong kyu-level players. This performance is obtained by a cascade of four relatively simple MLP classifiers in combination with a well-chosen representation, which only employs features that are calculated statically (without search).

By comparing numeric scores and counting unsettled interior points nearly all incorrectly scored final positions can be detected (for verification by a human operator). Although some final positions are assessed incorrectly by our classifier, most are in fact scored incorrectly by the players. Detecting games that were incorrectly scored by the players is important because most machine-learning methods require reliable training data for a good performance.

8.6.1 Future Work

By providing reliable score information CSA* opens the large source of Go knowledge which is implicitly available in human game records. The next step is to apply machine learning in non-final positions, which will be done in chapters 9 and 10. We believe that the representation, techniques, and the data set presented in this chapter provide a solid basis for static predictions in non-final positions.

So far, the good performance of CSA* was obtained without any search, indicating that static evaluation is sufficient for most human final positions. Nevertheless, we expect that some (selective) search can still improve the performance. Adding selective features that involve search and integrating our system into MAGOG, our 9×9 Go program, will be an important next step.

Although the performance of CSA* is already quite good for labelling game records, there are, at least in theory, still positions which may be scored incorrectly when the classifiers make the same mistakes as the human players. Future work should determine how often this happens in practice.

Chapter 9

Predicting life and death

This chapter is partially based on E. C. D. van der Werf, M. H. M. Winands, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to predict life and death from Go game records. *Information Sciences*, 2004. Accepted for publication. A 4-page paper summary appeared in [192].[1]

Over centuries humans have acquired extensive knowledge of Go. Much of this knowledge is implicitly available in the games of human experts. In the previous chapter, we have set the first step towards making the knowledge contained in 9×9 game records from the NNGS archive [136] accessible for machine-learning techniques. Consequently, we now have a database containing roughly 18,000 9 $\times 9$ games with reliable and complete score information. From this database we intend to learn relevant Go knowledge for building a strong evaluation function.

In this chapter we focus on predicting life and death. Unlike in the previous chapter, where we only used final positions, we now focus on predictions during the game. We believe that predicting life and death is a skill that is pivotal for strong play and an essential ingredient in any strong positional evaluation function.

The rest of this chapter is organised as follows. Section 9.1 introduces the concepts life and death. Section 9.2 presents the learning task in detail. In section 9.3 we discuss the representation which is extended with five additional features. Section 9.4 provides information on the data set. Section 9.5 reports on our experiments. Finally, section 9.6 gives the chapter conclusions.

9.1 Life and death

Life and death has been studied by various researchers [14, 45, 73, 100, 103, 112, 125, 127, 141, 176, 200]. It provides the basis for accurately evaluating Go positions. In this chapter we focus on learning to predict life and death for non-final positions from labelled game records. The labelling stored in the game

[1] The author would like to thank the editors of JCIS 2003, Elsevier Science, and his co-authors for permission to reuse relevant parts of the articles in this thesis.

records provides the controlling colour for each intersection at the end of the game.

The knowledge which blocks are dead and which are alive, at the end of a game, closely corresponds to the labelling of the intersections. Therefore, an intuitively straightforward implementation might be to learn to classify each block as the (majority of) occupied labelled points. However, this does not necessarily provide correct information for classification of life and death, for which at least two conflicting definitions exist.

The Japanese Go rules [135] state: "Stones are said to be 'alive' if they cannot be captured by the opponent, or if capturing them would enable a new stone to be played that the opponent could not capture. Stones which are not alive are said to be 'dead'."

The Chinese Go rules [54] state: "At the end of the game, stones which both players agree could inevitably be captured are dead. Stones that cannot be captured are alive."

(a) (b)

Figure 9.1: Alive or dead?

A consequence of both rules is shown in Figure 9.1a: the marked black stones can be considered alive by the Japanese rules, and dead by the Chinese rules. Since the white stones are dead under all rule sets, and the whole region is controlled by Black, the choice whether these black stones are alive or dead is irrelevant for scoring the position. However, whether the marked black stones should be considered alive or dead in training is unclear.

A more problematic position, known as '3 points without capturing', is shown in Figure 9.1b. If this position is scored under the Japanese rules all marked stones are considered alive (because after capturing some new stones would eventually be played that cannot be captured). However, if the position would be played out the most likely result (which may be different if one side can win a ko-fight) is that the empty point in the corner, the marked white stone, and the black stone marked with a triangle become black, and the three black stones marked with a square become white. Furthermore, all marked stones are captured and can therefore be considered dead under the Chinese rules.

In this chapter we choose the Chinese rules for defining life and death.

9.2 The learning task

By adopting the Chinese rules for defining life and death, the learning task becomes a task of predicting whether blocks of stones can or will be captured. In non-final positions, the blocks that *will* be captured (during the game) are easily recognised by looking ahead in the game record. Recognising the blocks that *can* be captured is more difficult.

In principle blocks that *can* be captured or saved may be recognised by goal-directed search. However, for the purpose of evaluating positions this may not be the best solution. The reason is that for some blocks, although they can be saved, the moves that would save them would constitute unreasonable play resulting in an unacceptable loss elsewhere on the board (meaning that optimal play would be to sacrifice such blocks). Conversely, capturing a block may also be unreasonable from a global perspective. Another difficulty is the inherent freedom of choice by the players. It is illustrated by the simple example in Figure 9.2. Here Black can capture one of the White blocks, while the other can be saved. Which of the two is captured, and which is saved is decided by the first player to play at point 'a' or 'b', the second player may then play at the other point. Consequently, it can be argued that the white blocks are 50% alive, and a prefect classification is therefore not possible.

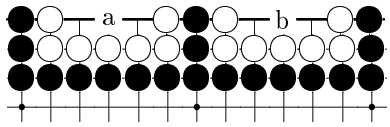


Figure 9.2: Fifty percent alive.

Since perfect classification is not possible in non-final positions, our goal is to approximate the Bayesian *a posteriori* probability given a set of features or at least the Bayesian discriminant function, for deciding whether the block will be alive or dead at the end of the game. This implicitly takes into account that play should be reasonable (or even optimal if the game record contains no mistakes). Moreover, we focus our attention only on blocks that will be relevant for scoring the positions at the end of the game. To approximate the Bayesian *a posteriori* probability we use the multi-layer perceptron (MLP) classifier. It has been shown [83] that minimising the mean-square error (MSE) on binary targets, for an MLP with sufficient functional capacity, adequately approximates the Bayesian *a posteriori* probability.

9.2.1 Target values for training

When replaying the game backward from the labelled final position the following four types of blocks are identified (in order of decreasing domination).

1. Blocks that are captured during the game.
2. Blocks that occupy points ultimately controlled by the opponent.
3. Blocks that occupy points on the edge of regions ultimately controlled by their own colour.

4. Blocks that occupy points in the interior of regions ultimately controlled by their own colour.

In contrast to normal play, when replaying the game backward blocks shrink and may split when stones are removed. When blocks shrink or split they inherit their type from the original block. Of course, when there is no change to a block the type is also preserved. When new blocks appear, because they were captured, they are marked type 1. Blocks of type 2, 3, and 4 obtained their initial labelling in the final position.

Blocks of type 1 and 2 should be classified as dead, and their target value for training is set to 0. Blocks of type 3 should be classified as alive, and their target value for training is set to 1. Type-4 blocks cannot be classified based on the labelling and are therefore not used in training. (As an example, the marked block in Figure 9.1a typically ends up as a type-4 block, and the marked blocks in Figure 9.1b as a type-2 block. However, if any of the marked blocks are actually captured during the game they will of course be of type 1.)

9.3 Five additional features

In chapter 8 we presented a representation for characterising blocks by several carefully selected features based on simple measurable geometric properties, some elementary Go knowledge, and some handcrafted specialised features. Since the representation performed quite well for final positions, we decided to re-use the same features for learning to predict life and death for non-final positions.

Of course, there are some features that are only relevant during the game. They were not used in chapter 8. We add the following five features.

- *Player to move* relative to the block's colour.
- *Ko* indicates if an active ko is on the board.
- *Distance to ko* from the block.
- *Number of friendly stones* on the board.
- *Number of opponent stones* on the board.

9.4 The data set

We used the same 9×9 game records played between 1995 and 2002 on NNGS [136] as in section 8.4. For the experiments reported in subsections 9.5.1 and 9.5.2 we used training and test examples obtained from 18,222 9×9 games that were played to the end and scored. In total, all positions from these games contain about 10 million blocks of which 8.5% are of type 1, 11.5% are of type 2, 65.5% are of type 3, and 14.5% are of type 4. Leaving out the type-4 blocks

gives as *a priori* probabilities that 76.5% of the remaining blocks are alive and 23.5% of the remaining blocks are dead.

In all experiments the test examples were extracted from games played in 1995, and the training examples from games played between 1996 and 2002. Since the games provide a huge amount of blocks with little or no variation (large regions remain unchanged per move) only a small fraction of blocks was randomly selected for training (<5% per game).

9.5 Experiments

This section reports on our experiments. In subsection 9.5.1 we start by choosing a classifier. Then, in 9.5.2 we measure the classifier performance over the game, and in 9.5.3 we present results on full-board evaluation of resigned games.

9.5.1 Choosing a classifier

It is important to choose a good classifier. In pattern recognition there is a variety of classifiers to choose from (see subsection 6.2.3). Our experiments in chapter 8 on scoring final positions showed that the multi-layer perceptron (MLP) provided a good performance with a reasonable training time. Consequently we decided to try the MLP on non-final positions too. There, the performance of the MLP mainly depended on the architecture, the number of training examples, and the training algorithm.

In the experiments reported here, we tested architectures with 1 and with 2 hidden layers containing various numbers of neurons per hidden layer. For training we compared: (1) gradient descent with momentum and adaptive learning (GDXNC) with (2) RPROP backpropagation (RPNC). For comparison we also present results for the nearest mean classifier (NMC), the linear discriminant classifier (LDC), and the logistic linear classifier (LOGLC) (see section 6.2.3).

In Table 9.1 the classifier performance is shown for a test set containing 22,632 blocks ($\sim$ 5%) extracted from 920 games played in 1995. The results are averages over 10 runs with different random weight initialisations. Training and validation sets were randomly selected per run (so in each run all classifiers used the same training data). The test set always remained fixed. The standard deviations are in the order of 0.1% for training with 25,000 examples, 0.2% for training with 5000 examples, and 0.5% for training with 1000 examples.

The results indicate that GDXNC performed slightly better than RPNC. Although RPNC trains 2 to 3 times faster than GDXNC, and converges at a lower error on the training data, the performance on the test data was slightly worse, probably because of overfitting. GDXNC-25 gave the best performance, classifying 88% of the blocks correctly. Using the double amount of neurons in the hidden layer of GDXNC-50 did not improve the performance. Adding a second hidden layer to the network architecture with 5 or 25 neurons (GDXNC-25-5, GDXNC-25-25) also did not improve the performance. Thus we may conclude that using one hidden layer with 25 neurons is sufficient at least for training

with 25,000 examples. Consequently, we selected the GDXNC-25 classifier for the experiments in the following sections.

Classifier	Test error (%)		
Training examples	1000	5000	25,000
NMC	21.5	21.0	21.0
LDC	14.2	13.7	13.6
LOGLC	14.8	13.3	13.1
GDXNC-5	13.8	12.9	12.2
GDXNC-15	13.9	12.9	12.2
GDXNC-25	**13.7**	**12.8**	**12.0**
GDXNC-25-5	13.8	12.9	12.1
GDXNC-25-25	13.9	**12.8**	**12.0**
GDXNC-50	13.8	**12.8**	**12.0**
RPNC-5	14.7	13.4	12.4
RPNC-15	14.4	13.2	12.4
RPNC-25	14.7	13.3	12.4
RPNC-25-5	14.3	13.4	12.6
RPNC-25-25	14.3	13.5	12.8
RPNC-50	15.0	13.3	12.5

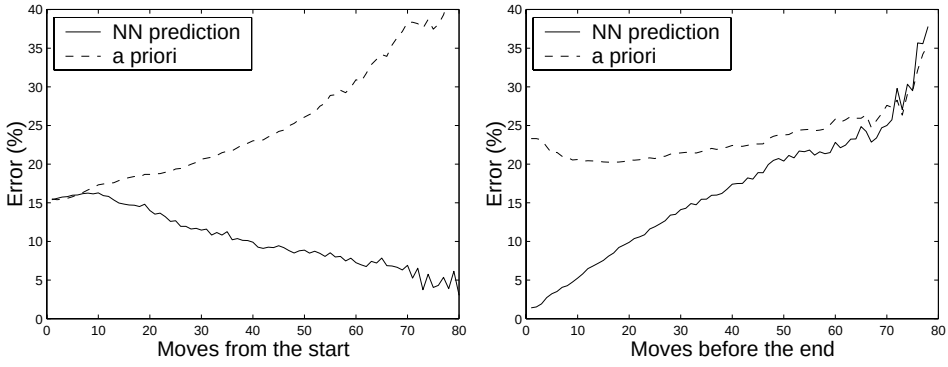
Table 9.1: Performance of classifiers. The numbers in the names indicate the number of neurons per hidden layer.

The performance of the classifiers strongly depends on the number of training examples. Therefore, we trained a new GDXNC-25 classifier on 175,000 examples. On average this classifier achieved a prediction error of 11.7% on the complete test set (containing 443,819 blocks from 920 games).

9.5.2 Performance during the game

In subsection 9.5.1 we calculated the average classification performance on blocks observed throughout the game. Consequently, the results in Table 9.1 do not tell us how the performance changes as the game develops. We hypothesise that for standard opening moves the best choice is pure guessing based on the highest *a priori* probability (always alive). Final positions, however, can (in principle) be classified perfectly. Given these extremes it is interesting to see how the performance changes over the game, either looking forward from the start position or backward from the final position.

To test the performance over the game we applied the GDXNC-25 classifier, trained on 175,000 examples, to all positions in the 920 test games and compared its performance to the *a priori* performance of always predicting alive. The performance looking forward from the start position is plotted in Figure 9.3a. The performance looking backward from the final position is plotted in Figure 9.3b.



(a) (b)

Figure 9.3: Performance over the game.

Figure 9.3a shows that pure guessing performs equally well for roughly the first 10 moves. As the length of games increases the *a priori* probability of blocks on the board ultimately being captured also increases (which makes sense because the best points are occupied first and there is only limited space on the board).[2]

Good evaluation functions typically aim at predicting the final result at the end of the game as soon as possible. It is therefore encouraging to see in Figure 9.3b that towards the end of the game the error goes down rapidly, predicting about 95% correctly 10 moves before the end. (For final positions over 99% of all blocks are classified correctly. The experiments in the previous chapter indicated that such a performance is at least comparable to that of the average rated 7-kyu NNGS player. Whether this performance is similar for non-final positions, far from the end of the game, is difficult to say.)

9.5.3 Full-board evaluation of resigned games

In the previous sections we only considered finished games that were played to the end and scored. However, not all games are finished. When humans observe that they are too far behind to win they usually resign. When games are resigned only the winner is stored in the game record. The life-and-death status of blocks is generally not available and may for some blocks even be unclear to human experts.

To test the performance of the GDXNC-25 classifier on resigned games it has to be incorporated into a full-board evaluation function that predicts the winner. In Go, full-board evaluation functions typically aim at predicting the number of intersections controlled by each player at the end of the game. The predictions of life and death, for the occupied intersections, provide the basis for

[2]We remark that although the plots extend only up to 80 moves this does not mean that there were no longer games. However, the number of games with a length of over 80 moves is too low for meaningful results.

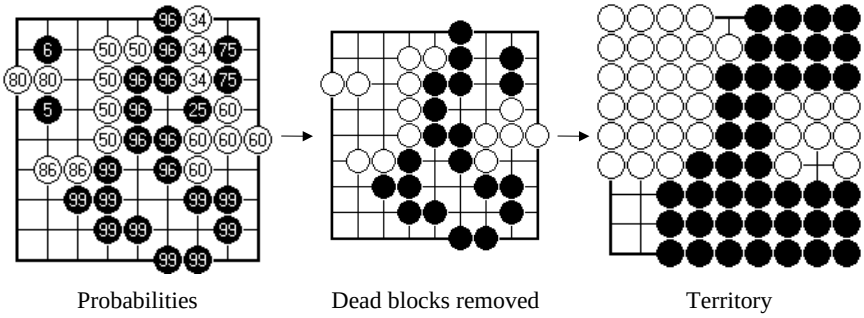


Figure 9.4: An example of a full-board evaluation.

such a full-board evaluation function. A straightforward extension[3] to classify all intersections is implemented by assigning each intersection to the colour of the nearest living block. An example is presented in Figure 9.4. Here the left board shows the predictions, the middle board shows all blocks which are assumed to be alive (with probability $\geq 50\%$), and the right board shows the territory which is calculated by assigning each intersection to the colour of the nearest living block. (Notice that even though one white dead block was incorrectly evaluated as 60% alive, the estimated territory is still sufficient to predict the correct winner.)

We tested 2,786 resigned 9×9 games played between 1995 and 2002 by rated players on NNGS [136]. On average the winner was predicted correctly for 87% of all positions. For comparison, if we do not remove any dead blocks, and all empty points are assigned to the colour of the nearest stone, the performance drops to 69% correct. This drop in performance underlines the importance of accurate predictions of life and death.

The strength of players is a factor that influences the difficulty of positions and the reliability of the results. We calculated statistics for all rank categories between 20 kyu and 2 dan. Figure 9.5 shows the relation between the rank of the player who resigned and the average error at predicting the winner (top), the number of game records available (middle), and the average estimated difference in points (bottom). The top plot suggests that predicting the winner tends to become more difficult with increasing strength of the players. This makes sense because strong players usually resign earlier (because they are better at recognising lost positions). Moreover, it may be that strong players generally create more difficult positions than weak players. It is no surprise that the estimated difference in points (when one player resigns) tends to decrease with the playing strength.

[3]More knowledgeable approaches will be discussed in the next chapter.

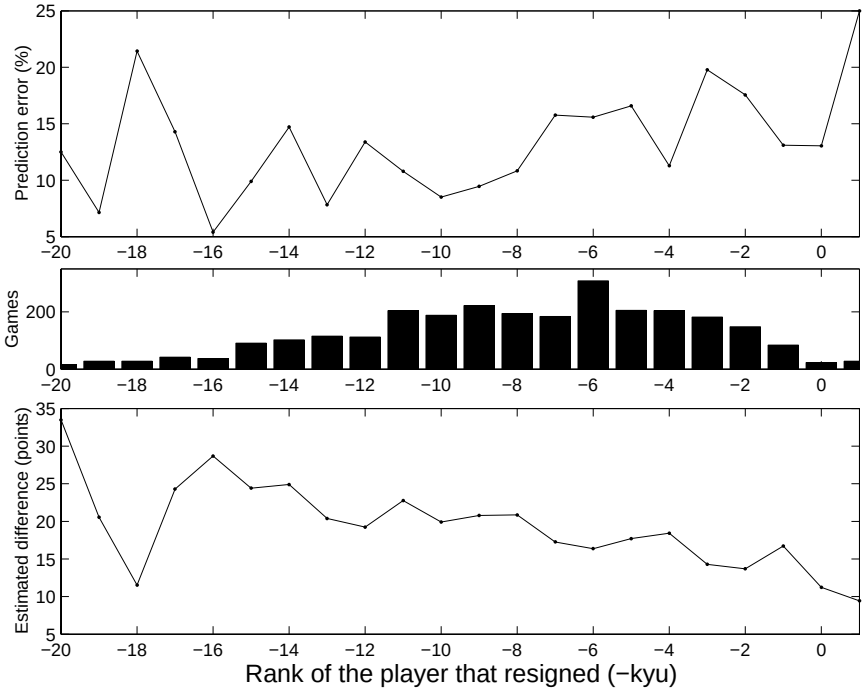


Figure 9.5: Predicting the outcome of resigned games.

9.6 Chapter conclusions

We trained MLPs to predict life and death from labelled examples quite accurately. From the experiments we may conclude that the GDXNC-25 classifier, which uses one hidden layer with 25 neurons, provides an adequate performance. Nevertheless, it should be noted that simple linear classifiers such as LOGLC also perform quite well. The reason for these similar performances probably lies in the quality of our representation, which helps to makes the classification task linearly separable.

On unseen game records and averaged over the whole game, the GDXNC-25 classifier classified around 88% of all blocks correctly. Ten moves before the end of the game it classified around 95% correctly, and for final positions it classified over 99% correctly.

We introduced a straightforward implementation of our GDXNC-25 classifier into a full-board evaluation function, which gave quite promising results. To obtain more insight into the importance of this work, the MLP should be incorporated into a more advanced full-board evaluation function. In chapter 10, this will be done for the task of estimating potential territory.

Regarding our second research question (see 1.3) we conclude that supervised learning techniques can be applied quite well for the task of predicting life and death in non-final positions. For positions near the end of the game we are

confident that the performance is comparable to that of reasonably strong kyu-level players. However, without additional experiments it is difficult to say whether the performance is similar in positions that are further away from the end of the game.

Future work

Although training with more examples still has some impact on the performance, it seems that most can be gained from improving the representation of blocks. Some features, such as those for loosely connected groups, have not yet been properly characterised and implemented. Adding selective features that involve search may also improve the performance. We conjecture that in the future automatic feature-extraction and feature-selection methods have to be employed to improve the representation.

Acknowledgements

We gratefully acknowledge financial support by the Universiteitsfonds Limburg / SWOL for presenting this work at JCIS 2003.

Chapter 10

Estimating potential territory

This chapter is based on E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to estimate potential territory in the game of Go. In *Proceedings of the 4th International Conference on Computers and Games (CG'04) (Ramat-Gan, Israel, July 5-7)*, 2004. To appear in LNCS, Springer-Verlag, Berlin, Germany.[1]

Evaluating Go positions is a difficult task [43, 127]. In the last decade Go has received significant attention from AI research [28, 126]. Yet, despite all efforts, the best Go programs are still weak. An important reason lies in the lack of an adequate full-board evaluation function. Building such a function requires a method for estimating potential territory. At the end of the game territory is defined as the intersections that are controlled by one colour. Together with the captured or remaining stones, territory determines who wins the game (see also subsection 2.2.4). For final positions (where both sides have completely sealed off the territory by stones of their colour) territory is determined by detecting and removing dead stones and assigning the empty intersections to their surrounding colour.

In chapter 8 we presented techniques for scoring final positions based on an accurate classification of life and death. In chapter 9 we extended our scope to predict life and death in non-final positions too. In this chapter we focus on evaluating non-final positions. In particular, we deal with the task of estimating potential territory in non-final positions. We believe that for this task predictions of life and death are a valuable component too. The current task is much more difficult than determining territory in final positions. We will investigate several possible methods to estimate potential territory based on the predictions of life and death and compare them to other approaches, known from the literature, which do not require an explicit notion of life and death.

[1] The author would like to thank Springer-Verlag and his co-authors for permission to reuse relevant parts of the article in this thesis.

The remainder of this chapter is organised as follows. First, in section 10.1 we define potential territory. Then, in section 10.2 we discuss five direct methods for estimating (potential) territory as well as two enhancements for supplying them with information about life and death. In section 10.3 we describe trainable methods for learning to estimate potential territory from examples. Section 10.4 presents our experimental setup. Then, in section 10.5 we present our experimental results. Finally, section 10.6 provides our chapter conclusions and suggestions for future research.

10.1 Defining potential territory

During the game human players typically try to estimate the territory that they will control at the end of the game. Moreover, they often distinguish between *secure territory*, which is assumed to be safe from attack, and *regions of influence*, which are unsafe. An important reason why human players like to distinguish secure territory from regions of influence is that, since the secure territory is assumed to be safe, they do not have to consider moves inside secure territory, which reduces the number of candidate moves to choose from.

In principle, secure territory can be recognised by extending Benson's method for recognising unconditional life [14], such as described in chapter 5 or in [125]. In practice, however, these methods are not sufficient to predict accurately the outcome of the game until the late end-game because they aim at 100% certainty, which is assured by assumptions like losing all ko-fights, allowing the opponent to place several moves without the defender answering, and requiring completely enclosed regions. Therefore, such methods usually leave too many points undecided.

An alternative (probably more realistic) model of the human notion of secure territory may be obtained by identifying regions with a high confidence level. However, finding a good threshold for distinguishing regions with a high confidence level from regions with a low confidence level is a non-trivial task and admittedly always a bit arbitrary. As a consequence it may be debatable to compare heuristic methods to methods with a 100% confidence level. Subsequently the debate continues when comparing among heuristic methods, e.g., a 77% versus a 93% confidence level (cf. Figure 10.1).

In this chapter, our main interest is in evaluating positions with the purpose of estimating the score. For this purpose the distinction between secure territory and regions of influence is relatively unimportant. Therefore we combine the two notions into one definition of *potential territory*.

Definition In a position, available from a game record, an intersection is defined as potential territory of a certain colour if the game record shows that the intersection is controlled by that colour at the end of the game.

Although it is not our main interest, it is possible to use our estimates of potential territory to provide a heuristic estimate of secure territory. This can be done by focusing on regions with a high confidence level, by setting an

arbitrarily high threshold. In subsection 10.5.3 we will present results at various levels of confidence so that our methods can be compared more extensively to methods that are designed for regions with a high confidence level only.

10.2 Direct methods for estimating territory

In this section we present five direct methods for estimating territory (subsections 10.2.1 to 10.2.5). They are known or derived from the literature and are easy to implement in a Go program. All methods assign a scalar value to each (empty) intersection. In general, positive values are used for intersections controlled by Black, and negative values for intersections controlled by White. In subsection 10.2.6 we mention two immediate enhancements for adding knowledge about life and death to the direct methods.

10.2.1 Explicit control

The explicit-control function is obtained from the 'concrete evaluation function' as described by Bouzy and Cazenave [28]. It is probably the simplest possible evaluation function and is included here as a baseline reference of performance. The explicit-control function assigns $+1$ to empty intersections which are completely surrounded by black stones and -1 to empty intersections which are completely surrounded by white stones, all other empty intersections are assigned 0.

10.2.2 Direct control

Since the explicit-control function only detects completely enclosed intersections (single-point eyes) as territory it performs quite weak. Therefore we propose a slight modification of the explicit-control function, called direct control. The direct-control function assigns $+1$ to empty intersections which are adjacent to a black stone and not adjacent to a white stone, -1 to empty intersections which are adjacent to a white stone and not adjacent to a black stone, and 0 to all other empty intersections.

10.2.3 Distance-based control

Both the explicit-control and the direct-control functions are not able to recognise larger regions surrounded by (loosely) connected stones. A possible alternative is the distance-based control (DBC) function. Distance-based control uses the Manhattan distance to assign $+1$ to each empty intersection that is closer to a black stone, -1 to each empty intersection that is closer to a white stone, and 0 to all other empty intersections.

10.2.4 Influence-based control

Although distance-based control is able to recognise larger territories a weakness is that it does not take into account the strength of stones in any way, i.e., a single stone is weighted equally important as a strong large block at the same distance. A way to overcome this weakness is by the use of influence functions, which were already described by the early researchers in computer Go Zobrist [203] and Ryder [151], and are still in use in several of today's Go programs [44, 47].

Below we adopt Zobrist's method to recognise influence; it works as follows. First, all intersections are initialised by one of three values: $+50$ if they are occupied by a black stone, -50 if they are occupied by a white stone, and 0 otherwise. (It should be noted that the value of 50 has no specific meaning and any other large value can be used in practice.) Then the following process is performed four times. For each intersection, add to the absolute value of the intersection the number of neighbouring intersections of the same sign minus the number of neighbouring intersections of the opposite sign.

10.2.5 Bouzy's method

It is important to note that the repeating process used to radiate the influence of stones in the Zobrist method is quite similar to the dilation operator known from mathematical morphology [163]. This was remarked by Bouzy [26] who proposed a numerical refinement of the classical dilation operator which is similar (but not identical) to Zobrist's dilation.

Bouzy's dilation operator D_z works as follows. For each non-zero intersection that is not adjacent to an intersection of the opposite sign, take the number of neighbouring intersections of the same sign and add it to the absolute value of the intersection. For each zero intersection without negative adjacent intersections, add the number of positive adjacent intersections. For each zero intersection without positive adjacent intersections, subtract the number of negative adjacent intersections.

Bouzy argued that dilations alone are not the best way to recognise territory. Therefore he suggested that the dilations should be followed by a number of erosions. This combined form is similar to the classical closing operator known from mathematical morphology [163].

To do this numerically Bouzy proposed the following refinement of the classical erosion operator E_z. For each non-zero intersection subtract from its absolute value the number of adjacent intersections which are zero or have the opposite sign. If this causes the value of the intersection to change its sign, the value becomes zero.

The operators E_z and D_z are then combined by first performing d times D_z followed by e times E_z, which we will refer to as $Bouzy(d, e)$. Bouzy suggested the relation $e = d(d-1)+1$ because this becomes the unity operator for a single stone in the centre of a sufficiently large board. He further recommended to use the values 4 or 5 for d. The intersections are initialised by one of three values:

+64 if they are occupied by a black stone, −64 if they are occupied by a white stone, and 0 otherwise.[2]

The reader may be curious why the number of erosions is larger than the number of dilations. The main reason is that (unlike in the classical binary case) Bouzy's dilation operator propagates faster than his erosion operator. Furthermore, Bouzy's method seems to be more aimed at recognising secure territory with a high confidence level than Zobrist's method (the intersections with a lower confidence level are removed by the erosions). Since Bouzy's method leaves many intersections undecided it is expected to perform sub-optimal at estimating potential territory, which also includes regions with lower confidence levels (cf. subsection 10.5.3). To improve the estimations of potential territory it is therefore interesting to consider an extension of Bouzy's method for dividing the remaining empty intersections. A natural choice to extend Bouzy's method is to divide the undecided empty intersections using distance-based control. The reason why we expect this combination to be better than only performing distance-based control directly from the raw board is that radiating influence from a (relatively) safe base, as provided by Bouzy's method, implicitly introduces some understanding of life and death. (It should be noted that extending Bouzy's method with distance-based control is not the only possible choice, and extending with, for example, influence-based control provides nearly identical results.)

10.2.6 Enhanced direct methods

The direct methods all share one important weakness: the lack of understanding life and death. As a consequence, dead stones (which are removed at the end of the game) can give the misleading impression of providing territory or reducing the opponent's territory. Recognising dead stones is a difficult task, but many Go programs have available some kind of (usually heuristic) information about the life-and-death status of stones. In our case we use the MLPs, trained to predict life and death for non-final positions, introduced in chapter 9.

Here we mention two immediate enhancements for the direct methods. (1) The simplest approach to use information about life and death for the estimation of territory is to remove dead stones before applying one of the direct methods. (2) An alternative sometimes used is to reverse the colour of dead stones [27].

10.3 Trainable methods

Although the direct methods can be improved by (1) removing dead stones, or (2) reversing their colour, neither approach seems optimal, especially because both lack the ability to exploit the more subtle differences in the strength of stones, which would be expressed by human concepts such as 'aji' or 'thickness'. However, since it is not well understood how such concepts should be modelled,

[2]We remark that these are the values proposed in Bouzy's original article [26]. For $d > 4$ larger initialisation values are required to prevent the possibility of removing single stones.

it is tempting to try a machine-learning approach to train a general function approximator to provide an estimation of the potential territory. For this task we again select the Multi-Layer Perceptron (MLP). The MLP has been used on similar tasks by several other researchers [51, 71, 72, 159], so we believe it is a reasonable choice. Nevertheless it should be clear that any other general function approximator can be used for the task.

Our MLP has a feed-forward architecture which estimates potential territory on a per intersection basis. The estimates are based on a local representation which includes features that are relevant for predicting the status of the intersection under investigation. Here we test two representations, first a simple one which only looks at the raw (local) configuration of stones, and second an enhanced representation that encompasses additional information about life and death.

For our experiments we exploit the fact that the game is played on a square board with eight symmetries. Furthermore, positions with Black to move are equal to positions with White to move provided that all stones reverse colour. To simplify the learning task we remove the symmetries in our representation by rotating the view on the intersection under investigation to one canonical region in the corner, and reversing the colours if the player to move is White.

10.3.1 The simple representation

The simple representation is characterised by the configuration of all stones in the region of interest (ROI) which is defined by all intersections within a predefined Manhattan distance of the intersection under investigation. For each intersection in the ROI we include the following feature:

- *Colour*: $+1$ if the intersection contains a black stone, -1 if the intersection contains a white stone, and 0 otherwise.

In the following, we will refer to the combination of the simple representation with an MLP trained to estimate potential territory as simple MLP (SMLP). The performance of the simple MLP will be compared to the direct methods because it does not use any explicit information of life and death (although some knowledge of life and death may of course be learned from examples) and only looks at the local configuration of stones. Since both Zobrist's and Bouzy's method (see above) are diameter limited by the number of times the dilation operator is used, our simple representation should be able to provide results which are at least comparable. However, we actually expect it to do better because the MLP might learn some additional shape-dependent properties.

10.3.2 The enhanced representation

We enhanced the simple representation with knowledge of life and death as provided by the GDXNC-25 classifier presented in the chapter 9. The most straightforward way to include the predictions of life and death would be to

add these predictions as an additional feature for each intersection in the ROI. However, preliminary experiments showed that this was not the best way to add knowledge of life and death. (The reason is that adding features reduces performance due to peaking phenomena caused by the curse of dimensionality [13, 92].) As an alternative which avoids increasing the dimensionality we decided to multiply the value of the colour feature in the simple representation with the estimated probability that the stones are alive. (This means that the sign of the value of an intersection indicates the colour, and the absolute value indicates some kind of strength.) Moreover, the following three features were added.

- *Edge:* encoded by a binary representation (board=0, edge=1) using a 9-bit string vector along the horizontal and vertical line from the intersection under investigation to the nearest edges.

- *Nearest colour:* the classification for the intersection using the distance-based control method on the raw board (black=1, empty=0, white=−1).

- *Nearest alive:* the classification for the intersection using the distance-based control method after removing dead stones (black=1, empty=0, white=−1).

In the following, the combination of the enhanced representation with an MLP trained to estimate potential territory is called enhanced MLP (EMLP).

10.4 Experimental setup

In this section we discuss the data set used for training and evaluation (subsection 10.4.1) and the performance measures used to evaluate the various methods (subsection 10.4.2).

10.4.1 The data set

In the experiments we used our collection of 9×9 game records which were originally obtained from NNGS [136]. The games, played between 1995 and 2002, were all played to the end and then scored. Since the original NNGS game records only contained a single numeric value for the score, the fate of all intersections was labelled by a threefold combination of GNU Go [74], our own learning system, and some manual labelling. Details about the data set and the way we labelled the games can be found in chapter 8

In all experiments, test examples were extracted from games played in 1995; training examples were extracted from games played between 1996 and 2002. In total the test set contained 906 games, 46,616 positions, and 2,538,152 empty intersections.

10.4.2 The performance measures

Now that we have introduced a series of methods (combinations of methods are possible too) to estimate (potential) territory, an important question is: how good are they? We attempt to answer this question (in section 10.5) using several measures of performance which can be calculated from labelled game records. Although game records are not ideal as an absolute measure of performance (because the people who played those games surely have made mistakes) we believe that the performance averaged over large numbers of unseen game records is a reasonable indication of strength.

Probably the most important question in assessing the quality of an evaluation function is how well it can predict the winner at the end of the game. By combining the estimated territory with the (alive) stones we obtain the so-called area score, which is the number of intersections controlled by Black minus the number of intersections controlled by White. Together with a possible komi (which compensates the advantage of the first player) the sign of this score determines the winner. Therefore, our first performance measure P_{winner} is the percentage of positions in which the sign of the score is predicted correctly.

Our second performance measure P_{score} uses the same score to calculate the average absolute difference between the predicted score and the actual score at the end of the game.

Both P_{winner} and P_{score} combine predictions of stones and territory in one measure of performance. As a consequence these measures are not sufficiently informative to evaluate the task of estimating potential territory alone. To provide more detailed information about the errors that are made by the various methods we also calculate the confusion matrices (see subsection 10.5.1) for the estimates of potential territory alone.

Since some methods leave more intersections undecided (i.e., by assigning empty) than others, it may seem unfair to compare them directly using only P_{winner} and P_{score}. As an alternative the fraction of intersections which are left undecided can be considered together with the performance on intersections which are decided. This typically leads to a trade-off curve where performance can be improved by rejecting intersections with a low confidence. The fraction of intersections that are left undecided, as well as the performance on the decided intersections is directly available from the confusion matrices of the various methods.

10.5 Experimental results

We tested the performance of the various direct and trainable methods. They are subdivided as follows: performance of direct methods in subsection 10.5.1; performance of trainable methods in subsection 10.5.2; comparing different levels of confidence in subsection 10.5.3; and performance during the game in subsection 10.5.4.

	P_{winner} (%)			P_{score} (points)		
Predicted dead stones	remain	remove	reverse	remain	remove	reverse
Explicit control	52.4	60.3	61.8	16.0	14.8	14.0
Direct control	54.7	66.5	66.9	**15.9**	12.9	12.7
Distance-based control	60.2	73.8	73.8	18.5	13.8	13.9
Influence-based control	61.0	73.6	73.6	17.3	12.8	12.9
Bouzy(4,13)	52.6	66.9	67.5	17.3	12.8	12.8
Bouzy(5,21)	55.5	70.2	70.4	17.0	**12.3**	**12.4**
Bouzy(5,21) + DBC	**63.4**	**73.9**	**73.9**	18.7	14.5	14.6

Table 10.1: Average performance of direct methods.

10.5.1 Performance of direct methods

The performance of the direct methods was tested on all positions from the labelled test games. The results for P_{winner} and P_{score} are shown in Table 10.1. In this table the columns 'remain' represent results without using knowledge of life and death, the columns 'remove' and 'reverse' represent results with predictions of life and death used to remove or reverse the colour of dead stones.

To compare the results of P_{winner} and P_{score} it is useful to have a confidence interval. However, since positions of the test set are not all independent, it is non-trivial to provide exact results. Nevertheless it is easy to calculate lower and upper bounds, based on an estimate of the number of independent positions. If we pessimistically assume only one independent position per game an upper bound (for a 95% confidence interval) is roughly 3% for P_{winner} and 1.2 points for P_{score}. If we optimistically assume all positions to be independent a lower bound is roughly 0.4% for P_{winner} and 0.2 points for P_{score}. Of course this is only a crude approximation which ignores the underlying distribution and the fact that the accuracy increases drastically towards the end of the game. However, given the fact that the average game length is around 50 moves it seems safe to assume that the true confidence interval will be somewhere in the order of 1% for P_{winner} and 0.4 points for P_{score}.

More detailed results about the estimations (in percentages) for the empty intersections alone are presented in the confusion matrices shown in Table 10.2. The fraction of undecided intersections and the performance on the decided intersections, which can be calculated from the confusion matrices, will be discussed in subsection 10.5.3. (The rows of the confusion matrices contain the possible predictions which are either black (PB), white (PW), or empty (PE). The columns contain the actual labelling at the end of the game which are either black (B), white (W), or empty (E). Therefore, correct predictions are found on the trace, and errors are found in the upper right and lower left corners of the matrices.)

The difference in performance between (1) when stones remain on the board and (2) when dead stones are removed or reversed colour underlines the impor-

	B	W	E
PB	0.78	0.16	0
PW	0.1	0.88	0
PE	48.6	49.3	0.13

Explicit control

	B	W	E
PB	0.74	0.04	0
PW	0.04	0.82	0
PE	48.7	49.5	0.14

dead stones removed

	B	W	E
PB	0.95	0.09	0.01
PW	0.07	1.2	0.01
PE	48.4	49.0	0.12

dead colour reversed

	B	W	E
PB	15.4	4.33	0.02
PW	3.16	14.3	0.01
PE	30.9	31.6	0.1

Direct control

	B	W	E
PB	16.0	3.32	0.02
PW	2.56	15.2	0.02
PE	30.9	31.7	0.09

dead stones removed

	B	W	E
PB	16.6	3.44	0.02
PW	2.75	16.3	0.03
PE	30.0	30.6	0.08

dead colour reversed

	B	W	E
PB	36.0	11.6	0.05
PW	6.63	31.0	0.03
PE	6.86	7.71	0.06

Distance-based control

	B	W	E
PB	38.2	10.4	0.06
PW	6.21	34.3	0.06
PE	5.09	5.55	0.02

dead stones removed

	B	W	E
PB	38.0	10.3	0.05
PW	6.21	34.1	0.05
PE	5.29	5.87	0.04

dead colour reversed

	B	W	E
PB	37.4	12.2	0.07
PW	7.76	33.3	0.04
PE	4.25	4.79	0.03

Influence-based control

	B	W	E
PB	38.4	10.5	0.06
PW	7.02	35.3	0.06
PE	4	4.47	0.02

dead stones removed

	B	W	E
PB	38.4	10.5	0.06
PW	7.11	35.4	0.06
PE	3.98	4.46	0.02

dead colour reversed

	B	W	E
PB	17.7	1.82	0.02
PW	0.83	15.4	0.01
PE	30.9	33.1	0.11

Bouzy(4,13)

	B	W	E
PB	21.2	1.86	0.03
PW	1.06	19.9	0.03
PE	27.2	28.5	0.07

dead stones removed

	B	W	E
PB	21.2	1.9	0.03
PW	1.16	20.1	0.03
PE	27.0	28.3	0.08

dead colour reversed

	B	W	E
PB	19.0	1.87	0.02
PW	0.81	15.8	0.01
PE	29.6	32.6	0.1

Bouzy(5,21)

	B	W	E
PB	23	1.98	0.03
PW	1.13	20.8	0.04
PE	25.3	27.5	0.07

dead stones removed

	B	W	E
PB	22.9	1.99	0.03
PW	1.17	20.7	0.03
PE	25.3	27.6	0.08

dead colour reversed

	B	W	E
PB	37.9	12.1	0.05
PW	6.51	32.4	0.03
PE	5	5.73	0.05

Bouzy(5,21) + DBC

	B	W	E
PB	39.0	11.0	0.06
PW	6.3	34.8	0.06
PE	4.12	4.5	0.02

dead stones removed

	B	W	E
PB	38.9	10.9	0.05
PW	6.32	34.6	0.05
PE	4.28	4.74	0.03

dead colour reversed

Table 10.2: Confusion matrices of direct methods.

	P_{winner} (%)			P_{score} (points)		
Predicted dead stones	remain	remove	reverse	remain	remove	reverse
Explicit control	55.0	66.2	68.2	16.4	14.5	13.3
Direct control	57.7	74.9	75.3	16.1	11.6	11.3
Distance-based control	61.9	82.1	82.1	16.4	9.6	9.7
Influence-based control	63.6	82.1	82.2	16.0	**9.5**	**9.6**
Bouzy(4,13)	56.7	77.9	78.3	17.2	10.4	10.5
Bouzy(5,21)	58.6	80.3	80.5	16.9	9.9	10
Bouzy(5,21) + DBC	**66.7**	**82.2**	**82.3**	**15.5**	9.6	9.7

Table 10.3: Average performance of direct methods after 20 moves.

tance of understanding life and death. For the weakest direct methods reversing the colour of dead stones seems to improve performance compared to only removing them. For the stronger methods, however, it has no significant effect.

The best method for predicting the winner without understanding life and death is Bouzy's method extended with distance-based control to divide the remaining undecided intersections. It is interesting to see that this method also has a high P_{score} which would actually indicate a bad performance. The reason for this is instability of distance-based control in the opening, e.g., with only one stone on the board it assigns the whole board to the colour of that stone. We can filter out the instability near the opening by only looking at positions that occur after a certain minimal number of moves. When we do this for all positions with at least 20 moves made, as shown in Table 10.3, it becomes clear that Bouzy's method extended with distance-based control also achieves the best P_{score}. Our experiments indicate that radiating influence from a (relatively) safe base, as provided by Bouzy's method, outperforms other direct methods probably because it implicitly introduces some understanding of life and death. This conclusion is supported by the observation that the combination does not perform significantly better than for example influence-based control when knowledge about life and death is used.

At first glance the results presented in this subsection could lead to the tentative conclusion that for a method which only performs N dilations to estimate potential territory the performance keeps increasing with N; so the largest possible N might have the best performance. However, this is not the case and N should not be chosen too large. Especially in the beginning of the game a large N tends to perform significantly worse than a restricted setting with 4 or 5 dilations such as used by Zobrist's method. Moreover, setting N too large leads to a waste of time under tournament conditions.

10.5.2 Performance of trainable methods

Below we present the results of the trainable methods. All architectures were trained with the resilient propagation algorithm (RPROP) developed by Ried-

miller and Braun [146]. The non-linear architectures all had one hidden layer with 25 units using the hyperbolic tangent sigmoid transfer function. (Preliminary experiments showed this to be a reasonable setting, though large networks may still provide a slightly better performance when more training examples are used.) For training, 200,000 examples were used. A validation set of 25,000 examples was used to stop training. For each architecture the weights were trained three times with different random initialisations, after which the best result was selected according to the performance on the validation set. (Note that the validation examples were taken, too, from games played between 1996 and 2002.)

We tested the various linear and non-linear architectures on all positions from the labelled test games. Results for P_{winner} and P_{score} are presented in Table 10.4, and the confusion matrices are shown in Table 10.5. The enhanced representation, which uses predictions of life and death, clearly performs much better than the simple representation. We further see that the performance tends to improve with increasing size of the ROI. (A ROI of size 24, 40, and 60 corresponds to the number of intersections within a Manhattan distance of 3, 4, and 5 respectively, excluding the centre point, which is always empty.)

Architecture	Representation	ROI	P_{winner} (%)	P_{score} (points)
linear	simple	24	64.0	17.9
linear	simple	40	64.5	18.4
linear	simple	60	64.6	19.0
non-linear	simple	24	63.1	18.2
non-linear	simple	40	64.5	18.3
non-linear	simple	60	65.1	18.3
linear	enhanced	24	75.0	13.4
linear	enhanced	40	75.2	13.3
linear	enhanced	60	75.1	13.4
non-linear	enhanced	24	75.2	13.2
non-linear	enhanced	40	**75.5**	12.9
non-linear	enhanced	60	**75.5**	**12.5**

Table 10.4: Performance of the trainable methods.

It is interesting to see that the non-linear architectures are not much better than the linear architectures. This seems to indicate that, once life and death has been established, influence spreads mostly linearly.

10.5.3 Comparing different levels of confidence

The MLPs are trained to predict positive values for black territory and negative values for white territory. Small values close to zero indicate that intersections are undecided and by adjusting the size of the window around zero, in which

	B	W	E
PB	40.5	13.6	0.08
PW	6.7	33.8	0.05
PE	2.3	2.9	0.01

Simple, linear, roi=24

	B	W	E
PB	41.5	14.2	0.08
PW	5.8	33.2	0.05
PE	2.1	2.9	0.01

Simple, linear, roi=40

	B	W	E
PB	42.0	14.6	0.08
PW	5.3	32.6	0.04
PE	2.2	3.1	0.01

Simple, linear, roi=60

	B	W	E
PB	40.5	13.6	0.07
PW	6.4	33.3	0.05
PE	2.6	3.5	0.02

Simple, non-linear, roi=24

	B	W	E
PB	41.4	14.0	0.08
PW	5.8	33.0	0.04
PE	2.4	3.3	0.02

Simple, non-linear, roi=40

	B	W	E
PB	41.8	14.2	0.0759
PW	5.5	33.0	0.04
PE	2.2	3.2	0.01

Simple, non-linear, roi=60

	B	W	E
PB	40.5	11.7	0.06
PW	7.0	36.4	0.07
PE	2.0	2.3	0.01

Enhanced, linear, roi=24

	B	W	E
PB	40.6	11.6	0.06
PW	6.8	36.3	0.07
PE	2.1	2.5	0.01

Enhanced, linear, roi=40

	B	W	E
PB	40.6	11.6	0.06
PW	6.6	36.2	0.07
PE	2.2	2.6	0.01

Enhanced, linear, roi=60

	B	W	E
PB	40.3	11.4	0.06
PW	6.8	36.2	0.06
PE	2.4	2.7	0.01

Enhanced, non-linear, roi=24

	B	W	E
PB	40.4	11.3	0.06
PW	6.8	36.4	0.06
PE	2.4	2.7	0.01

Enhanced, non-linear, roi=40

	B	W	E
PB	40.2	10.9	0.06
PW	6.8	36.6	0.07
PE	2.5	2.9	0.01

Enhanced, non-linear, roi=60

Table 10.5: Confusion matrices of trainable methods.

we predict empty, we can modify the confidence level of the non-empty classifications. If we do this we can plot a trade-off curve which shows how the performance increases at the cost of rejecting undecided intersections.

In Figure 10.1 two such trade-off curves are shown for the simple MLP and the enhanced MLP, both non-linear with a ROI of size 60. For comparison, results for the various direct methods are also plotted. It is shown that the MLPs perform well at all levels of confidence. Moreover, it is interesting to see that at high confidence levels Bouzy(5,21) performs nearly as good as the MLPs.

Although Bouzy's methods and the influence methods provide numerical results, which could be used to plot trade-off curves, too, we did not do this because they would make the plot less readable. Moreover, for Bouzy's methods the lines would be quite short and uninteresting because they already start high.

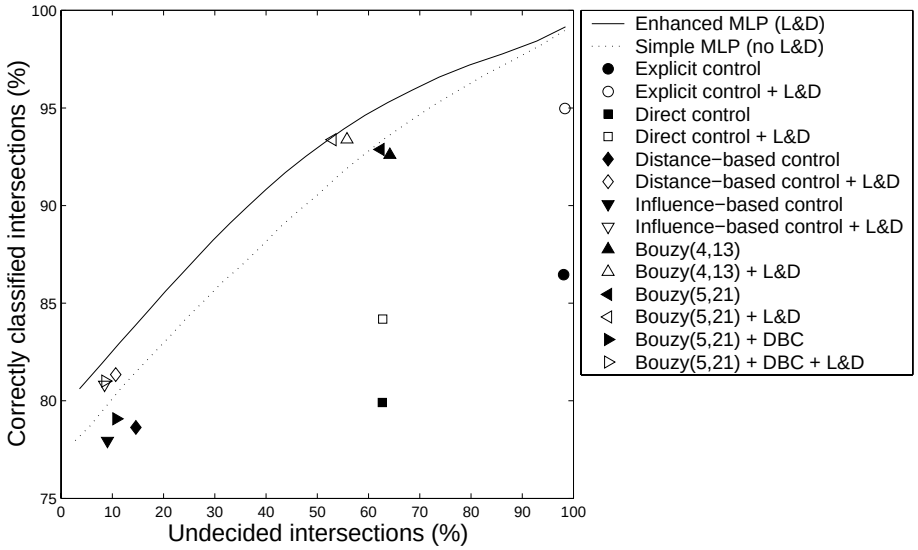


Figure 10.1: Performance at different levels of confidence.

10.5.4 Performance during the game

In the previous subsections we looked at the average performance over complete games. Although this is interesting, it does not tell us how the performance changes as the game develops. Below we consider the performance changes and the adequacy of the MLP performance.

Since all games do not have equal length, there are two principal ways of looking at the performance. First, we can look forward from the start, and second, we can look backward from the end. The results for P_{winner} are shown in Figure 10.2a looking forward from the start and in Figure 10.2b looking backward from the end. We remark that the plotted points are between moves and their associated performance is the average obtained for the two directly adjacent positions (where one position has Black to move and the other has White to move). This was done to filter out some distracting odd-even effects caused by the alternation of the player to move. It is shown that the EMLP, using the enhanced representation, performs best. However, close to the end Bouzy's method extended with distance-based control and predictions of life and death performs nearly as good. The results for P_{score} are shown in Figure 10.2c looking forward from the start and in Figure 10.2d looking backward from the end. Also here we see that the EMLP performs best.

For clarity of presentation we did not plot the performance of DBC, which is rather similar to Influence-based control (IBC) (but over-all slightly worse). For the same reason we did not plot the results for DBC and IBC with knowledge of life and death, which perform quite similar to Bouzy(5,21)+DBC+L&D.

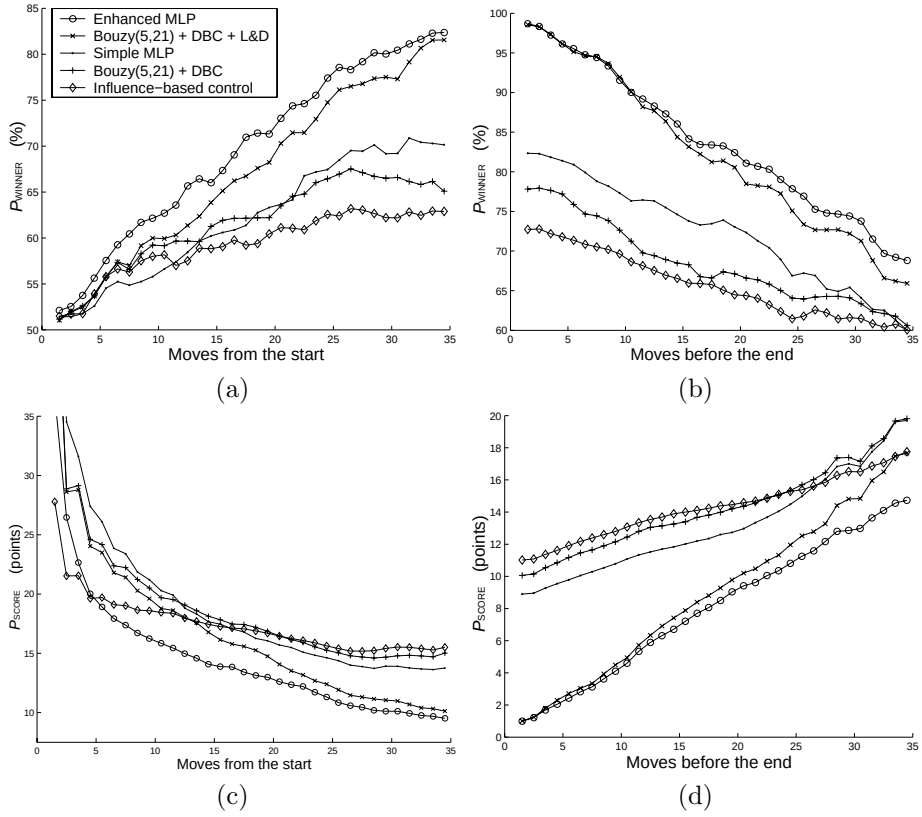


Figure 10.2: Performance over the game.

It is interesting to observe how good the simple MLP performs. It outperforms all direct methods without using life and death. Here it should be noted that the adequate performance of the simple MLP could still be improved considerably, if it would be allowed to make predictions for occupied intersections too, i.e., remove dead stones. (This was not done for a fair comparison with the direct methods.)

10.6 Chapter conclusions

We investigated several direct and trainable methods for estimating potential territory. We tested the performance of the direct methods, known from the literature, which do not require an explicit notion of life and death. Additionally, two enhancements for adding knowledge of life and death and an extension of Bouzy's method were presented.

From the experiments we may conclude that without explicit knowledge of life and death the best direct method is Bouzy's method extended with distance-

based control to divide the remaining empty intersections. If information about life and death is used to remove dead stones this method also performs well. However, the difference with distance-based control and influence-based control becomes small.

Moreover, we presented new trainable methods for estimating potential territory. They can be used in combination with the classifiers for predicting life and death presented in chapter 9. Using only the simple representation the SMLP can estimate potential territory at a level outperforming the best direct methods. The EMLP, which has the knowledge of life and death available from the GDXNC-25 classifier presented in chapter 9, performs well at all stages of the game, even at high levels of confidence. Experiments showed that all methods are greatly improved by adding knowledge of life and death, which leads us to conclude that good predictions of life and death are the most important ingredient for an adequate full-board evaluation function.

Regarding our second research question (see 1.3) we conclude that supervised learning techniques can be applied quite well for the task of estimating potential territory. When provided with sufficient training examples, these techniques easily outperform the best direct methods known from literature. On a human scale, we are confident that for positions near the end of the game the performance is at least comparable to that of reasonably strong kyu-level players. However, as in chapter 9, without additional experiments it is difficult to say whether the performance is similar in positions that are far away from the end of the game.

Future research

Although our system for predicting life and death already performs quite well, we believe that it can still be improved significantly. The most important reason is that we only use static features, which do not require search. By incorporating features from specialised life-and-death searches the predictions of life and death may improve. By improving the predictions of life and death, the performance of the EMLP should improve as well.

The work in chapter 8 indicated that CSA* scales up well to the 19×19 board. Although we expect similar results for the EMLP, additional experiments should be performed to validate this claim.

In our experiments we estimated potential territory based on knowledge extracted from game records. An interesting alternative for acquiring such knowledge may be obtaining it by simulation using, e.g., Monte Carlo methods [29, 34].

Acknowledgements

We gratefully acknowledge financial support by the Universiteitsfonds Limburg / SWOL for presenting this work at CG 2004.

Chapter 11

Conclusions and future research

In the thesis we investigated AI techniques for the game of Go. We focused our research on searching techniques and on learning techniques. We combined the techniques with adequate knowledge representations, and presented practical implementations that show how they can be used to improve the strength of Go programs.

We developed search-based programs for the capture game (Ponnuki) and for Go (Migos) that can solve the game and play perfectly on small boards. Moreover, we developed learning systems for move prediction (MP*), scoring final positions (CSA*), predicting life and death (GDXNC-25), and estimating potential territory (EMLP). The various techniques have all been implemented in the Go program Magog which has won the bronze medal in the 9×9 Go tournament of the 2004 Computer Olympiad in Ramat-Gan, Israel.

The course of this final chapter is as follows. In section 11.1 we provide answers to the research questions, and summarise the main conclusions of the individual chapters. Then, in section 11.2 we return to the problem statement of section 1.3. Finally, in section 11.3 we present some directions for future research.

11.1 Answers to the research questions

In section 1.3 we posed the following two research questions.

1. *To what extent can searching techniques be used in computer Go?*

2. *To what extent can learning techniques be used in computer Go?*

In the following two subsections we answer these questions.

11.1.1 Searching techniques

To address the first research question we summarise the main conclusions of chapters 4 and 5. They focused on searching techniques. Thereafter we give our main conclusion.

The capture game: PONNUKI

Our program PONNUKI solved the capture game on empty square boards up to size 5×5. The 6×6 board is solved, too, under the assumption that the first four moves are played in the centre. These results were obtained by a combination of standard searching techniques, some standard enhancements adapted to exploit domain-specific properties of the game, and a novel evaluation function.

We conclude that standard searching techniques and enhancements can be applied successfully for the capture game, especially when they are restricted to small regions of fewer than 30 empty intersections. Moreover, we conclude that our evaluation function performs adequately at least for the task of capturing stones.

Solving Go on small boards: MIGOS

The main result is that MIGOS, as the first Go program in the world, solved Go on the 5×5 board. Further, the program solved several 6×6 positions with 8 and more stones on the board. The results were reached by a combination of standard search enhancements (transposition tables, enhanced transposition cut-offs), improved search enhancements (history heuristic, killer moves, sibling promotion), and new search enhancements (internal unconditional bounds, symmetry lookups), a dedicated heuristic evaluation function, and a method for static recognition of unconditional territory.

So far, only the 4×4 board was solved by Sei and Kawashima [161]. For this board their search required 14,000,000 nodes. MIGOS was able to confirm their solutions and solved the same board in fewer than 700,000 nodes. Hence we conclude that the static recognition of unconditional territory, the symmetry lookups, the enhanced move ordering, and our Go-specific improvements to the various search enhancements are key ingredients for solving Go on small boards.

We analysed the application of the situational-super-ko rule (SSK) and identified several problems that can occur due to the graph-history-interaction problem [36] in tree search using transposition tables. Most problems can be overcome by only allowing transpositions that are found at the same depth in the search tree (transpositions found at a different depth can, of course, still be used for the move ordering). The remaining problems are quite rare, especially in combination with a decent move ordering, and can often be ignored safely because winning scores under basic ko are lower bounds on the score under SSK. 这是为啥？

We conclude that, on current hardware, provably correct solutions can be obtained within a reasonable time frame for confined regions of size up to about 28 intersections. Moreover, for efficiency of the search, provably correct domain-specific knowledge is essential to obtain tight bounds on the score early in the

search tree. Without such domain-specific knowledge, detecting final positions by search alone becomes unreasonably expensive.

Conclusion

We applied searching techniques in two domains with a reduced complexity. By scaling the complexity both domains provided an interesting testing ground for investigating the limitations of the current state of the art in searching techniques. The experimental results showed that advances in search enhancements, provably correct knowledge for the evaluation function, and the ever increasing computing power drastically increase the power of searching techniques. When searches are confined to regions of about 20 to 30 intersections, the current state-of-the-art searching techniques together with adequate domain-specific knowledge representations can provide strong and often even perfect play.

11.1.2 Learning techniques

To address the second research question we summarise the main conclusions of chapters 7, 8, 9, and 10, which focused on learning techniques. It is followed by our main conclusion.

Move prediction: MP*

We have presented a system that learns to predict moves in the game of Go from observing strong human play. The performance of our best move predictor (MP*) is, at least in local regions, comparable to that of strong kyu-level players.

The training algorithm presented here is more efficient than standard fixed-target implementations. This is mainly due to the avoidance of needless weight adaptation when rankings are correct. As an extra bonus, our training method reduces the number of gradient calculations as the performance grows, thus speeding up the training. A major contribution to the performance is the use of feature-extraction methods. Feature extraction reduces the training time while increasing the quality of the predictor. Together with a sensible scaling of the original features and an optional second-phase training, superior performance over direct-training schemes can be obtained.

The predictor can be used for move ordering and forward pruning in a full-board search. The performance obtained on ranking professional moves indicates that a large fraction of the legal moves may be pruned directly. In particular, our results against GNU Go indicate that a relatively small set of high-ranked moves is sufficient to play a strong game.

We conclude that it is possible to train a learning system to predict good moves most of the time with a performance at least comparable to strong kyu-level players. The performance was obtained from a simple set of locally computable features, thus ignoring a significant amount of information which can be obtained by more extensive (full-board) analysis or by specific goal-directed

searches. Consequently, there is still significant room for improving the performance, possibly even into the strong dan-level region.

Scoring final positions: CSA*

We developed a Cascaded Scoring Architecture (CSA*) that learns to score final positions from labelled examples. On unseen game records our system scored around 98.9% of the positions correctly without any human intervention. Compared to the average rated player on NNGS, who has a rating of 7 kyu for scored 9×9 games, we may conclude that CSA* is more accurate at removing all dead blocks, and performs comparably on determining the correct winner.

We conclude that for the task of scoring final positions supervised learning techniques can provide a performance at least comparable to reasonably strong kyu-level players. This performance can be obtained by a cascade of relatively simple MLP classifiers in combination with a well-chosen representation, which only employs features that are calculated statically (without search).

By comparing numeric scores and counting unsettled interior points nearly all incorrectly scored final positions can be detected (for verification by a human operator). Although some final positions are assessed incorrectly by our classifier, most are in fact scored incorrectly by the players. Detecting games that were incorrectly scored by the players is important for obtaining reliable training data.

Predicting life and death: GDXNC-25

We trained MLPs to predict life and death from labelled examples quite accurately. From the experiments we may conclude that the GDXNC-25 classifier provides an adequate performance. Nevertheless, it should be noted that simple linear classifiers such as LOGLC also perform quite well. The reason probably lies in the quality of our representation, which helps to make the classification task linearly separable.

On unseen game records and averaged over the whole game, the GDXNC-25 classifier classified around 88% of all blocks correctly. Ten moves before the end of the game it classified around 95% correctly, and for final positions it classified over 99% correctly.

We conclude that supervised learning techniques can be applied quite well for the task of predicting life and death in non-final positions. For positions near the end of the game we are confident that the performance is comparable to that of reasonably strong kyu-level players. However, without additional experiments it is difficult to say whether the performance is similar in positions that are further away from the end of the game.

Estimating potential territory: EMLP

We investigated several direct and trainable methods for estimating potential territory. We tested the performance of the direct methods, known from the

literature, which do not require an explicit notion of life and death. Additionally, two enhancements for adding knowledge of life and death and an extension of Bouzy's method were presented.

From the experiments we may conclude that without explicit knowledge of life and death the best direct method is Bouzy's method extended with distance-based control to divide the remaining empty intersections. When information about life and death is used to remove dead stones the difference with distance-based control and influence-based control becomes small, and all three methods perform quite well.

Moreover, we presented new trainable methods for estimating potential territory. They can be used as an extension of our system for predicting life and death. Using only a simple representation our trainable methods can estimate potential territory at a level outperforming the best direct methods. Experiments showed that all methods are greatly improved by adding knowledge of life and death, which leads us to conclude that good predictions of life and death are the most important ingredient for an adequate full-board evaluation function.

We conclude that supervised learning techniques can be applied quite well for the task of estimating potential territory. When provided with sufficient training examples, these techniques easily outperform the best direct methods known from literature. On a human scale, we are confident that for positions near the end of the game the performance is at least comparable to that of reasonably strong kyu-level players. However, without additional experiments it is difficult to say whether the performance is similar in positions that are far away from the end of the game.

Conclusion

We applied learning techniques on important Go-related tasks and compared the performance with the performance of human players as well as with other programs and techniques. We showed that learning techniques can effectively extract knowledge for evaluating moves and positions from human game records. The most important ingredients for obtaining a good performance from learning techniques are (1) carefully chosen representations, and (2) large amounts of reliable training data. The quality of the static knowledge that can be obtained using learning techniques is at least comparable to that of reasonably strong kyu-level players.

11.2 Answer to the problem statement

In section 1.3 we formulated the following problem statement:

> *How can AI techniques be used to improve the strength of Go programs?*

To answer this question we investigated AI techniques for searching and for learning in the game of Go.

Our conclusion is that domain-specific knowledge is the most important ingredient for improving the strength of Go programs. For small problems, human experts can implement sufficient provably correct knowledge. Larger problems require heuristic knowledge.

Although human experts also can implement heuristic knowledge, this tends to become quite difficult when the playing strength of the program increases. To overcome this problem learning techniques can be used to automatically extract knowledge from the game records of human experts. For maximising performance, the main task of the programmer then becomes providing the learning algorithms with adequate representations and large amounts of reliable training data. Once adequate knowledge is available, searching techniques can exploit the ever increasing computing power to improve the strength of Go programs further. In addition, some goal-directed search may also be used to improve the representation.

11.3 Directions for future research

When studying the AI techniques for searching and for learning, on specific tasks in relative isolation, we observed that they could significantly improve the strength of Go programs. The searching techniques generally work best for well-defined tasks with a restricted number of intersections involved. As the number of involved intersections grows, heuristic knowledge becomes increasingly important. The learning techniques can provide this knowledge, and consequently, learning techniques are most important for dealing with the more fuzzy tasks typically required to play well on large boards.

To build a strong Go program the techniques should be combined. Partially this has already been done in our Go program MAGOG, which has recently won the bronze medal (after finishing 4th in 2003) in a strong field of 9 participants in the 9×9 Go tournament of the 2004 Computer Olympiad in Ramat-Gan, Israel. However, many of the design choices in MAGOG were made ad hoc and the tuning between the various components is probably still far from optimal.

Four combinations of the techniques developed in this thesis are directly apparent:

1. goal-directed capture searches can be used to provide features for improving the representation for predicting life and death,

2. the move predictor can be used for move ordering or even forward pruning in a tree-searching algorithm,

3. the heuristic knowledge for evaluating positions (chapter 10) can be combined with the provably correct knowledge and search of MIGOS,

4. the predictions of life and death and the estimations of territory can be used to improve the representation for move prediction.

Next to the combinations of techniques there are several other possible directions for future research. Below, we mention some ideas that we consider most interesting.

The capture game

For the capture game the next challenges are: solving the empty 6×6 board and solving the 8×8 board starting with a crosscut in the centre. It would be interesting to compare the performance with other searching techniques such as the various PN-searching techniques [3, 4, 31, 130, 162, 197].

Solving Go on small boards

The next challenges in small-board Go are: solving the 6×6 and 7×7 boards. Both boards are claimed to have been solved by humans, but so far no computer was able to confirm the results.

We expect large gains from adding more (provably correct) Go knowledge to the evaluation function (for obtaining final scores earlier in the tree). Further, a scheme for selective search extensions, examining a highly asymmetric tree (resembling the human solutions), may enable the search to solve the 6×6 and the 7×7 board much more efficiently than fixed-depth iterative-deepening search without extensions. Next to these suggestions an improved move ordering may increase the search efficiency, possibly even by several orders of magnitude.

As for the capture game, it would be interesting to compare the performance of MIGOS with the various alternative PN-searching techniques [3, 4, 31, 130, 162, 197].

Reinforcement learning

In this thesis we mainly focused on supervised learning. Although we have presented some preliminary work on reinforcement learning (see section 6.4), and even had some interesting results recently for small-board Go [67, 68], there still remains much work to be done. When the techniques developed in this thesis are fully integrated in a Go program, it will be interesting to train them further, by self-play and by playing against other opponents, using reinforcement learning.

Move prediction

For move prediction, it may be interesting to train the move predictor further through some type of Q-learning. In principle Q-learning works regardless of who is selecting the moves. Consequently, training should work both by self-play (in odd positions) and by replaying human games. Since Q-learning does not rely on the assumption of the optimality of human moves, it may be able to solve possible inconsistencies in its current knowledge (due to the fact that some human moves were bad). Moreover, it may be well suited to explore lines of play that are never considered by human players.

Scoring final positions

The performance of CSA* is already quite good for the purpose of labelling large quantities of game records, particularly because most incorrectly scored positions are easily detected and can be presented for human inspection. However, for automating the process further we could still add more static features as well as non-static features that could use some (selective) search. In addition, automatic play-out by a full Go-playing engine may be an interesting alternative for dealing with the most difficult positions.

Predicting life and death

For predicting life and death during the game, we believe most can be gained by improving the representation of blocks. Some features, such as those for loosely connected groups, have not yet been implemented, whereas some other features may be correlated or could even be redundant. We expect that adding features that involve some (selective) search may improve the performance considerably. Most likely, automatic feature-extraction and feature-selection methods have to be employed to improve the representation.

Estimating potential territory

We expect that the best way to improve the performance of the EMLP is by improving the predictions of life and death.

References

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–52, 1994.

[2] S. G. Akl and M. M. Newborn. The principal continuation and the killer heuristic. In *1977 ACM Annual Conference Proceedings*, pages 466–473. ACM Press, New York, NY, 1977.

[3] L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. Ph.D. thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands, 1994.

[4] L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

[5] A. G. Arkedev and E. M. Braverman. *Computers and Pattern Recognition*. Thomson, Washington, DC, 1966.

[6] A. Aÿ. Frühe erwähnungen des Go in Europa, 2004. http://go-lpz.envy.nu/vorgeschichte.html

[7] L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kauffman, San Francisco, CA, 1995.

[8] E. B. Baum and W. D. Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97:195–242, 1997.

[9] J. Baxter, A. Tridgell, and L. Weaver. Tdleaf(λ): Combining temporal difference learning with game-tree search. *Australian Journal of Intelligent Information Processing Systems*, 5(1):39–43, 1998.

[10] D. F. Beal. Learn from you opponent - but what if he/she/it knows less than you? In J. Retschitzki and R. Haddad-Zubel, editors, *Step by Step. Proceedings of the 4th Colloquium Board Games in Academia*, pages 123–132. Editions Universitaires, Fribourg, Switzerland, 2002.

[11] D. F. Beal and M. C. Smith. Learning piece values using temporal difference learning. *ICCA Journal*, 20(3):147–151, 1997.

[12] R. C. Bell. *Board and Table Games from Many Civilizations*. Oxford University Press, 1960. Revised edition, 1979.

[13] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, NJ, 1961.

[14] D. B. Benson. Life in the game of Go. *Information Sciences*, 10:17–29, 1976. Reprinted in D.N.L Levy, editor, *Computer Games*, Vol. II, pages 203–213, Springer-Verlag, New York, NY, 1988.

[15] H. J. Berliner. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.

[16] H. J. Berliner, G. Goetsch, M. S. Campbell, and C. Ebeling. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43(1):7–20, 1990.

[17] H. J. Berliner and C. McConnell. B* probability based search. *Artificial Intelligence*, 86(1):97–156, 1996.

[18] D. Billings, L. Peña, J. Schaeffer, and D. Szafron. Learning to play strong poker. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 11, pages 225–242. Nova Science Publishers, Huntington, NY, 2001.

[19] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, UK, 1995.

[20] Y. Björnsson. *Selective Depth-First Game-Tree Search*. Ph.D. thesis, University of Alberta, Edmonton, Canada, 2002.

[21] Y. Björnsson and T. A. Marsland. Multi-cut $\alpha\beta$-pruning in game-tree search. *Theoretical Computer Science*, 252:177–196, 2001.

[22] Y. Björnsson and T. A. Marsland. Learning extension parameters in game-tree search. *Information Sciences*, 154(3-4):95–118, 2003.

[23] A. Blair. Emergent intelligence for the game of Go. In *From Animals to Animats, The 6th International Conference on the Simulation of Adaptive Behavior (SAB2000)*. ISAB, MA, 2000.

[24] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1-3):235–282, 1989.

[25] B. Bouzy. Spatial reasoning in the game of Go, 1996. http://www.math-info.univ-paris5.fr/∼bouzy/publications/SRGo.article.pdf

[26] B. Bouzy. Mathematical morphology applied to computer Go. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(2):257–268, 2003.

[27] B. Bouzy, 2004. Personal communication.

[28] B. Bouzy and T. Cazenave. Computer Go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–102, 2001.

[29] B. Bouzy and B. Helmstetter. Monte-Carlo Go developments. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 159–174. Kluwer Academic Publishers, Boston, MA, 2003.

[30] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 369–376. The MIT Press, Cambridge, MA, 1995.

[31] D. M. Breuker. *Memory versus Search in Games*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands, 1998.

[32] D. M. Breuker, J. W. H. M. Uiterwijk, and H. J. van den Herik. Replacement schemes and two-level tables. *ICCA Journal*, 19(3):175–180, 1996.

[33] British Go Association. Comparison of some Go rules, 2001. http://www.britgo.org/rules/compare.html

[34] B. Brügmann. Monte Carlo Go, 1993. ftp://ftp.cse.cuhk.edu.hk/pub/neuro/GO/mcgo.tex

[35] M. Buro. Toward opening book learning. *ICCA Journal*, 22(2):98–102, 1999.

[36] M. Campbell. The graph-history interaction: on ignoring position history. In *Proceedings of the 1985 ACM Annual Conference on the Range of Computing: Mid-80's Perspective*, pages 278–280. ACM Press, New York, NY, 1985.

[37] D. Carmel and S. Markovitch. Learning models of opponent's strategy in game playing. Technical Report CIS Report 9318, Technion - Israel Institute of Technology, Computer Science Department, Haifa, Israel, 1993.

[38] T. Cazenave. Automatic acquisition of tactical Go rules. In H. Matsubara, editor, *Proceedings of the 3rd Game Programming Workshop*, Hakone, Japan, 1996.

[39] T. Cazenave. Metaprogramming forced moves. In H. Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 645–649, Brighton, U.K., 1998.

[40] T. Cazenave, 2002. Personal communication.

[41] T. Cazenave. Gradual abstract proof search. *ICGA Journal*, 25(1):3–16, 2002.

[42] T. Cazenave. La recherche abstraite graduelle de preuve. *13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle*, 8 - 10 Janvier 2002, Centre des Congrès d'Angers, 2002. http://www.ai.univ-paris8.fr/∼cazenave/AGPS-RFIA.pdf

[43] K-H. Chen. Some practical techniques for global search in Go. *ICGA Journal*, 23(2):67–74, 2000.

[44] K-H. Chen. Computer Go: Knowledge, search, and move decision. *ICGA Journal*, 24(4):203–215, 2001.

[45] K-H. Chen and Z. Chen. Static analysis of life and death in the game of Go. *Information Sciences*, 121:113–134, 1999.

[46] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[47] Z. Chen. Semi-empirical quantitative theory of Go part i: Estimation of the influence of a wall. *ICGA Journal*, 25(4):211–218, 2002.

[48] J. Churchill, R. Cant, and D. Al-Dabass. Genetic search techniques for line of play generation in the game of Go. In Q. H. Mehdi, N. E. Gough, and S. Natkine, editors, *Proceedings of GAME-ON 2003 4th International Conference on Intelligent Games and Simulation*, pages 233–237. EUROSIS, 2003.

[49] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J. John, editor, *International Conference on Genetic Algorithms and their Applications (ICGA85)*. Carnegie Mellon University, Pittsburgh, PA, 1985.

[50] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, UK, 2000.

[51] F. A. Dahl. Honte, a Go-playing program using neural nets. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 10, pages 205–223. Nova Science Publishers, Huntington, NY, 2001.

[52] J. Davies. Small-board problems. *Go World*, 14-16:55–56, 1979.

[53] J. Davies. Go in lilliput. *Go World*, 17:55–56, 1980.

[54] J. Davies. The rules of Go. In R. Bozulich, editor, *The Go Player's Almanac*. Ishi Press, San Francisco, CA, 1992. http://www-2.cs.cmu.edu/∼wjh/go/rules/Chinese.html

[55] J. Davies. 5x5 Go. *American Go Journal*, 28(2):9–12, 1994.

[56] J. Davies. 5x5 Go revisited. *American Go Journal*, 29(3):13, 1995.

[57] J. Davies. 7x7 Go. *American Go Journal*, 29(3):11, 1995.

[58] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[59] T. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 12*, pages 994–1000, 2000.

[60] H. H. L. M. Donkers. *Nosce Hostem - Searching with Opponent Models*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2003.

[61] C. Donninger. Null move and deep search. *ICCA Journal*, 16(3):137–143, 1993.

[62] T. Drange. Mini-Go, 2002. http://www.mathpuzzle.com/go.html

[63] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. 2nd edition. Wiley, New York, NY, 2001.

[64] R. P. W. Duin. Compactness and complexity of pattern recognition problems. In C. Perneel, editor, *Proc. Int. Symposium on Pattern Recognition 'In Memoriam Pierre Devijver' (Brussels, B, Feb.12)*, pages 124–128. Royal Military Academy, Brussels, 1999.

[65] R. P. W. Duin. *PRTools, a Matlab Toolbox for Pattern Recognition*. Pattern Recognition Group, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands, 2000.

[66] D. Dyer. Searches, tree pruning and tree ordering in Go. In H. Matsubara, editor, *Proceedings of the Game Programming Workshop in Japan '95*, pages 207–216. Computer Shogi Association, Tokyo, 1995.

[67] R. Ekker. Reinforcement learning and games. M.Sc. thesis, RijksUniversiteit Groningen, Groningen, 2003.

[68] R. Ekker, E. C. D. van der Werf, and L. R. B. Schomaker. Dedicated TD-learning for stronger gameplay: applications to Go. In A. Nowé, T. Lennaerts, and K. Steenhaut, editors, *Proceedings of Benelearn 2004 Annual Machine Learning Conference of Belgium and The Netherlands*, pages 46–52. Vrije Universiteit Brussel, Brussel, Belgium, 2004.

[69] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[70] H. D. Enderton. The Golem Go program. Technical Report CMU-CS-92-101, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1991.

[71] M. Enzenberger. The integration of a priori knowledge into a Go playing neural network, 1996. http://www.markus-enzenberger.de/neurogo1996.html

[72] M. Enzenberger. Evaluation in Go by a neural network using soft segmentation. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 97–108. Kluwer Academic Publishers, Boston, MA, 2003.

[73] D. Fotland. Static eye analysis in 'The Many Faces of Go'. *ICGA Journal*, 25(4):201–210, 2002.

[74] Free Software Foundation. GNU Go, 2004. http://www.gnu.org/software/gnugo/gnugo.html

[75] K. Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition*. Academic Press, New York, NY, 1990.

[76] J. Fürnkranz. Machine learning in games: A survey. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 2, pages 11–59. Nova Science Publishers, Huntington, NY, 2001.

[77] F. A. Gers. *Long Short-Term Memory in Recurrent Neural Networks*. Ph.D. thesis, Department of Computer Science, Swiss Federal Institute of Technology, EPFL, Lausanne, Switzerland, 2001.

[78] W. Gerstner and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Cambridge, UK, 2002.

[79] S. B. Gray. Local properties of binary images in two dimensions. *IEEE Transactions on Computers*, C-20(5):551–561, 1971.

[80] K. Gurney. *An Introduction to Neural Networks*. UCL Press, London, UK, 1997.

[81] M. Hagan, H. Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, Boston, MA, 1996.

[82] M. Hagan and M. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

[83] J. B. Hampshire II and B. A. Pearlmutter. Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function. In D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*, pages 159–172. Morgan Kaufmann, San Mateo, CA, 1990.

[84] E. A. Heinz. *Scalable Search in Computer Chess*. Vieweg Verlag, Braunschweig, Germany, 2000.

[85] H. J. van den Herik, H. Iida, and E. A. Heinz, editors. *Advances in Computer Games: Many Games, Many Challenges*. Kluwer Academic Publishers, Boston, MA, 2004.

[86] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.

[87] J. Hertz, A. S. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, Redwood City, CA, 1990.

[88] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[89] R. M. Hyatt. Book learning - a methodology to tune an opening book automatically. *ICCA Journal*, 22(1):3–12, 1999.

[90] IGS. The internet Go server, 2002. http://igs.joyjoy.net/

[91] H. Iida, J. W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg. Potential applications of opponent-model search. part 1: The domain of applicability. *ICCA Journal*, 16(4):201–208, 1993.

[92] A. Jain and B. Chandrasekaran. Dimensionality and sample size considerations in pattern recognition practice. In P. R. Krishnaiah and L. N. Kanal, editors, *Handbook of Statistics*, volume 2, pages 835–855. North-Holland, Amsterdam, The Netherlands, 1982.

[93] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[94] R. Jasiek. Commentary on the nihon kiin 1989 rules, 1997. http://home.snafu.de/jasiek/j1989com.html

[95] R. Jasiek, 2004. Personal communication.

[96] I. T. Jollife. *Principal Component Analysis*. Springer-Verlag, New York, NY, 1986.

[97] A. Junghanns and J. Schaeffer. Search versus knowledge in game-playing programs revisited. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 692–697. Morgan Kaufmann, San Francisco, CA, 1997.

[98] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[99] J. Karhunen, E. Oja, L. Wang, R. Vigario, and J. Joutsensalo. A class of neural networks for independant component analysis. *IEEE Transactions on Neural Networks*, 8(3):486–504, 1997.

[100] A. Kishimoto and M. Müller. Df-pn in Go: An application to the one-eye problem. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 125–141. Kluwer Academic Publishers, Boston, MA, 2003.

[101] A. Kishimoto and M. Müller. A general solution to the graph history interaction problem. In *Nineteenth National Conference on Artificial Intelligence (AAAI'04)*, pages 644–649. AAAI Press, 2004.

[102] A. Kishimoto and M. Müller. A solution to the GHI problem for depth-first proof-number search. *Information Sciences*, 2004. Accepted for publication.

[103] T. Klinger. *Adversarial Reasoning: A Logical Approach for Computer Go*. Ph.D. thesis, New York University, New York, 2001.

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

[105] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[106] L. Kocsis. *Learning Search Decisions*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2003.

[107] L. Kocsis, J. W. H. M. Uiterwijk, E. O. Postma, and H. J. van den Herik. The neural movemap heuristic in chess. In J. Schaeffer, M. Müller, and Y. Björnsson,

editors, *Computers and Games: Third International Conference, CG 2002, Edmonton, Canada, July 2002: revised papers*, volume 2883 of *LNCS*, pages 154–170. Springer-Verlag, Berlin, Germany, 2003.

[108] T. Kohonen. *Self-organising Maps*. Springer-Verlag, Berlin, Germany, 1995.

[109] T. Kojima. *Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches*. Ph.D. thesis, University of Tokyo, Tokyo, Japan, 1998.

[110] G. Konidaris, D. Shell, and N. Oren. Evolving neural networks for the capture game. In *Proceedings of the SAICSIT Postgraduate Symposium*. Port Elizabeth, South Africa, 2002.

[111] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[112] H. A. Landman. Eyespace values in Go. In R. Nowakowski, editor, *Games of No Chance*, pages 227–257. Cambridge University Press, Cambridge, UK, 1996.

[113] E. Lasker. *Go and Go-Moku*. Dover Publications Inc., New York, NY, 1960. Dover edition, revised version of the work published in 1934 by Alfred A. Knopf.

[114] G. W. Leibniz. Annotatio de quibusdam ludis. In *Miscellanea Berolinensia Ad Incrementum Scientiarum*. Ex Scriptis Societati Regiae Scientiarum, Berlin, Germany, 1710.

[115] D. Lichtenstein and M. Sipser. Go is polynomial-space hard. *Journal of the ACM*, 27(2):393–401, 1980.

[116] T. R. Lincke. Strategies for the automatic construction of opening books. In T. A. Marsland and I. Frank, editors, *Computers and Games: Proceedings of the 2nd International Conference, CG 2000*, volume 2063 of *LNCS*, pages 74–86. Springer-Verlag, Berlin, Germany, 2001.

[117] D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

[118] T. A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.

[119] G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York NY, 2000.

[120] M. L. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969. Expanded Edition, 1988.

[121] D. E. Moriarty and R. Miikkulainen. Evolving neural networks to focus minimax search. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pages 1371–1377. AAAI/MIT Press, Cambridge, MA, 1994.

[122] D. E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[123] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

[124] M. Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. Ph.D. thesis, ETH Zürich, Zürich, Switzerland, 1995.

[125] M. Müller. Playing it safe: Recognizing secure territories in computer Go by using static rules and search. In H. Matsubara, editor, *Proceedings of the Game Programming Workshop in Japan '97*, pages 80–86. Computer Shogi Association, Tokyo, Japan, 1997.

[126] M. Müller. Computer Go. *Artificial Intelligence*, 134(1-2):145–179, 2002.

[127] M. Müller. Position evaluation in computer Go. *ICGA Journal*, 25(4):219–228, 2002.

[128] M. Müller, 2003. Personal communication.

[129] A. Nagai. A new and/or tree search algorithm using proof number and disproof number. In *Proceedings Complex Games Lab Workshop, ETL*, pages 40–45. Tsukuba, Japan, 1998.

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

[131] D. S. Nau. Pathology on game trees: A summary of results. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 102–104, 1980.

[132] H. L. Nelson. Hash tables in Cray Blitz. *ICCA Journal*, 8(1):3–13, 1985.

[133] J. von Neumann. Zur Theorie der Gesellschaftspiele. *Mathematische Annalen*, 100:295–320, 1928.

[134] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 2nd edition, 1947.

[135] Nihon Kiin and Kansai Kiin. The Japanese rules of Go, 1989. Translated by J. Davies, diagrams by J. Cano, reformatted, adapted, and edited by F. Hansen. http://www-2.cs.cmu.edu/∼wjh/go/rules/Japanese.html

[136] NNGS. The no name Go server game archive, 2002. http://nngs.cosmic.org/gamesearch.html

[137] B.A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

[138] J. Peng and R. Williams. Incremental multi-step q-learning. *Machine Learning*, 22:283–290, 1996.

[139] A. Plaat. *Research Re: search & Re-search*. Ph.D. thesis, Tinbergen Institute and Department of Computer Science, Erasmus University Rotterdam, Rotterdam, 1996.

[140] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Exploiting graph properties of game trees. In *Proceedings of the Thirteenth National Conference on Articial Intelligence (AAAI'96)*, volume 1, pages 234–239, 1996.

[141] R. Popma and L. V. Allis. Life and death refined. In H. J. van den Herik and L. V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3*, pages 157–164. Ellis Horwood, London, 1992.

[142] M. Pratola and T. Wolf. Optimizing GOTOOLS search heuristics using genetic algorithms. *ICGA Journal*, 26(1):28–48, 2003.

[143] A. Reinefeld. An improvement to the scout tree search algorithm. *ICCA Journal*, 6(4):4–14, 1983.

[144] H. Remus. Simulation of a learning machine for playing Go. In *IFIP Congress 1962*, pages 428–432. North-Holland Publishing Company, 1962. Reprinted in D.N.L Levy, editor, *Computer Games*, Vol. II, pages 136–142, Springer-Verlag, New York, 1988.

[145] N. Richards, D. E. Moriarty, and R. Miikkulainen. Evolving neural networks to play Go. *Applied Intelligence*, 8(1):85–96, 1998.

[146] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation: the RPROP algorithm. In H. Rusini, editor, *Proceedings of the IEEE Int. Conf. on Neural Networks (ICNN)*, pages 586–591, 1993.

[147] J. M. Robson. The complexity of Go. In *IFIP World Computer Congress 1983*, pages 413–417, 1983.

[148] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan press, Washington, DC, 1962.

[149] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–363. MIT Press, Cambridge MA, 1986.

[150] P. Rutquist. Evolving an evaluation function to play Go. Stage d'option de l'École Polytechnique, France, 2000.
http://www.eeaax.polytechnique.fr/papers/theses/per.ps.gz

[151] J. L. Ryder. *Heuristic Analysis of Large Trees as Generated in the Game of Go*. Ph.D. thesis, Stanford University, Stanford, CA, 1971.

[152] J. W. Sammon, Jr. A non-linear mapping for data structure analysis. *IEEE Transactions on Computers*, 18:401–409, 1969.

[153] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[154] A. L. Samuel. Some studies in machine learning using the game of checkers ii -recent progress. *IBM Journal of Research and Development*, 11:601–617, 1967.

[155] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[156] J. Schaeffer. *Experiments in Search and Knowledge*. Ph.D. thesis, Department of Computing Science, University of Waterloo, Waterloo, Canada, 1986.

[157] J. Schaeffer, M. Müller, and Y. Björnsson, editors. *Computers and Games: Third International Conference, CG 2002, Edmonton, Canada, July 2002: revised papers*, volume 2883 of *LNCS*. Springer-Verlag, Berlin, Germany, 2003.

[158] J. Schaeffer and A. Plaat. Kasparov versus DEEP BLUE: The rematch. *ICCA Journal*, 20(2):95–101, 1997.

[159] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski. Temporal difference learning of position evaluation in the game of Go. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing 6*, pages 817–824. Morgan Kaufmann, San Francisco, CA, 1994.

[160] R. Schütze. Atarigo 1.0, 1998. Free to download at the site of the DGoB (Deutscher Go-Bund): http://www.dgob.de/down/index_down.htm

[161] S. Sei and T. Kawashima. A solution of Go on 4x4 board by game tree search program. In *The 4th Game Informatics Group Meeting in IPS Japan*, pages 69–76 (in Japanese), 2000. http://homepage1.nifty.com/Ike/katsunari/paper/4x4e.txt

[162] M. Seo, H. Iida, and J. W. H. M. Uiterwijk. The pn*-search algorithm: Application to tsume-shogi. *Artificial Intelligence*, 129:253–277, 2001.

[163] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[164] D. J. Slate and L. R. Atkin. CHESS 4.5 - the northwestern university chess program. In P. W. Frey, editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag, New York, NY, 1977.

[165] W. L. Spight, 2003. Personal communication.

[166] P. H. M. Spronck, I. G. Sprinkhuizen-Kuyper, and E. O. Postma. Evolving improved opponent intelligence. In Q. Mehdi, N. Gough, and M. Cavazza, editors, *Proceedings of GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 94–98. SCS Europe Bvba, Ghent, Belgium, 2002.

[167] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[168] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, A Bradford Book, Cambridge, MA, 1998.

[169] G. Tesauro. Connectionist learning of expert preferences by comparison training. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 1 (NIPS-88)*, pages 99–106. Morgan Kaufmann, San Francisco, CA, 1989.

[170] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[171] D. Torrieri. The eigenspace separation transform for neural-network classifiers. *Neural Networks*, 12(3):419–427, 1999.

[172] J. Tromp, 2002. Personal communication.

[173] J. N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[174] Y. Tsuruoka, D. Yokoyama, and T. Chikayama. Game-tree search algorithm based on realization probability. *ICGA Journal*, 25(3):145–152, 2002.

[175] J. W. H. M. Uiterwijk and H. J. van den Herik. The advantage of the initiative. *Information Sciences*, 122(1):43–58, 2000.

[176] R. Vilà and T. Cazenave. When one eye is sufficient: a static classification. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 109–124. Kluwer Academic Publishers, Boston, MA, 2003.

[177] C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.

[178] N. Wedd. Goemate wins Go tournament. *ICCA Journal*, 23(3):175–177, 2000.

[179] P. J. Werbos. Backpropagation through time; what it does and how to do it. In *Proceedings of the IEEE*, volume 78, pages 1550–1560, 1990.

[180] E. C. D. van der Werf. Non-linear target based feature extraction by diabolo networks. M.Sc. thesis, Pattern Recognition Group, Department of Applied Physics, Delft University of Technology, Delft, The Netherlands, 1999.

[181] E. C. D. van der Werf. Aya wins 9×9 Go tournament. *ICCA Journal*, 26(4):263, 2003.

[182] E. C. D. van der Werf, J. W. H. M. Uiterwijk, E. O. Postma, and H. J. van den Herik. Local move prediction in Go. In J. Schaeffer, M. Müller, and Y. Björnsson, editors, *Computers and Games: Third International Conference, CG 2002, Edmonton, Canada, July 2002: revised papers*, volume 2883 of *LNCS*, pages 393–412. Springer-Verlag, Berlin, Germany, 2003.

[183] E. C. D. van der Werf, J. W. H. M. Uiterwijk, and H. J. van den Herik. Learning connectedness in binary images. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'01)*, pages 459–466. Universiteit van Amsterdam, Amsterdam, The Netherlands, 2001.

[184] E. C. D. van der Werf, J. W. H. M. Uiterwijk, and H. J. van den Herik. Programming a computer to play and solve Ponnuki-Go. In Q. Mehdi, N. Gough, and M. Cavazza, editors, *Proceedings of GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 173–177. SCS Europe Bvba, Ghent, Belgium, 2002.

[185] E. C. D. van der Werf, J. W. H. M. Uiterwijk, and H. J. van den Herik. Solving Ponnuki-Go on small boards. In H. Blockeel and M. Denecker, editors, *Proceedings of 14th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'02)*, pages 347–354. K.U.Leuven, Leuven, Belgium, 2002.

[186] E. C. D. van der Werf, J. W. H. M. Uiterwijk, and H. J. van den Herik. Solving Ponnuki-Go on small boards. In J. W. H. M. Uiterwijk, editor, *7th Computer Olympiad Computer-Games Workshop Proceedings*. Technical Report CS 02-03, IKAT, Department of Computer Science, Universiteit Maastricht, Maastricht, The Netherlands, 2002.

[187] E. C. D. van der Werf and H. J. van den Herik. Visual learning in Go. In J.W.H.M. Uiterwijk, editor, *The CMG 6th Computer Olympiad: Computer-Games Workshop Proceedings Maastricht*. Technical Report CS 01-04, IKAT, Department of Computer Science, Universiteit Maastricht, Maastricht, The Netherlands, 2001.

[188] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to score final positions in the game of Go. In H. J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games: Many Games, Many Challenges*, pages 143–158. Kluwer Academic Publishers, Boston, MA, 2003.

[189] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Solving Go on small boards. *ICGA Journal*, 26(2):92–107, 2003.

[190] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to estimate potential territory in the game of Go. In *Proceedings of the 4th International Conference on Computers and Games (CG'04) (Ramat-Gan, Israel, July 5-7)*, 2004. To appear in LNCS, Springer-Verlag, Berlin, Germany.

[191] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to score final positions in the game of Go. *Theoretical Computer Science*, 2004. Accepted for publication.

[192] E. C. D. van der Werf, M. H. M. Winands, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to predict life and death from Go game records. In Ken Chen

et al., editor, *Proceedings of JCIS 2003 7th Joint Conference on Information Sciences*, pages 501–504. JCIS/Association for Intelligent Machinery, Inc., 2003.

[193] E. C. D. van der Werf, M. H. M. Winands, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to predict life and death from Go game records. *Information Sciences*, 2004. Accepted for publication.

[194] M. A. Wiering. TD learning of game evaluation functions with hierarchical neural architectures. M.Sc. thesis, University of Amsterdam, Amsterdam, The Netherlands, 1995.

[195] M. A. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.

[196] M. H. M. Winands, L. Kocsis, J. W. H. M. Uiterwijk, and H. J. van den Herik. Temporal difference learning and the neural movemap heuristic in the game of Lines of Action. In Q. Mehdi, N. Gough, and M. Cavazza, editors, *Proceedings of GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation*, pages 99–103. SCS Europe Bvba, Ghent, Belgium, 2002.

[197] M. H. M. Winands, J. W. H. M. Uiterwijk, and H. J. van den Herik. An effective two-level proof-number search algorithm. *Theoretical Computer Science*, 313:511–525, 2004.

[198] M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, and E. C. D. van der Werf. Enhanced forward pruning. In *Proceedings of JCIS 2003 7th Joint Conference on Information Sciences*, pages 485–488, Research Triangle Park, North Carolina, USA, 2003. JCIS/Association for Intelligent Machinery, Inc.

[199] M. H. M. Winands, E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. The relative history heuristic. In *Proceedings of the 4th International Conference on Computers and Games (CG'04) (Bar-Ilan University, Ramat-Gan, Israel, July 5-7)*, 2004. To appear in LNCS, Springer-Verlag, Berlin, Germany.

[200] T. Wolf. The program gotools and its computer-generated tsume go database. In *1st Game Programming Workshop in Japan*, Hakone, Japan, 1994.

[201] X. Yao. Evolving artificial neural networks. *PIEEE: Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[202] R. Zaman and D. C. Wunsch II. TD methods applied to mixture of experts for learning 9x9 Go evaluation function. In *Proceedings of the 1999 International Joint Conference on Neural Networks (IJCNN '99)*, volume 6, pages 3734–3739. IEEE, 1999.

[203] A. L. Zobrist. A model of visual organization for the game Go. In *Proceedings of AFIPS 1969 Spring Joint Computer Conference*, volume 34, pages 103–112. AFIPS Press, Montvale, NJ, 1969.

[204] A. L. Zobrist. *Feature Extraction and Representation for Pattern Recognition and the Game of Go*. Ph.D. thesis, University of Wisconsin, Madison, WI, 1970.

[205] A. L. Zobrist. A new hashing method with application for game playing. Technical Report 88, Computer Science Department, University of Wisconsin, Madison, WI, 1970. Reprinted (1990) in *ICCA Journal*, 13(2):69-73.

Appendix A

MIGOS rules

This appendix provides a precise formulation of rules which enables a computer to play and solve Go while avoiding problems that may occur due to some ambiguities of the traditional rules. Our main aim is to approximate traditional area-scoring rules without super ko as close as possible. In the footnotes we present some optional changes/additions that (usually) make the search more efficient, but can have some rare (and usually insignificant) counter-intuitive side effects.

We remark that the MIGOS rules are a special-purpose formulation of the rules for programming computer Go. If anyone would wish to adopt these rules for human play they should at least be extended with the option of agreement about life and death in the scoring phase, and the option of resigning.

A.1 General

The game of Go is played by two players, Black and White, on a rectangular grid (usually 19×19). Each intersection of the grid is coloured black if it contains a black stone, white if it contains a white stone, or empty if it contains no stone. One player uses black stones, the other white stones. Initially the board is empty. The player with the black stones starts the game. The players move alternately. A move is either a play of a stone on an empty intersection, or a pass.

A.2 Connectivity and liberties

Two intersections are adjacent if they have a line but no intersection between them. Two adjacent intersections are connected if they have the same colour. Two non-adjacent intersections are connected if there is a path of adjacent intersections of their colour between them. Connected intersections of the same colour form a block. An intersection that is not connected to any other inter-

section is also a block. The directly adjacent empty intersections of blocks are called liberties.

A block of stones is captured when the opponent plays on its last liberty. When stones are captured they are removed from the grid.

A.3 Illegal moves

Basic ko: A move may not remove a single stone if this stone has removed a single stone in the last preceding move.

Suicide: A move that does not capture an opponent block and leaves its own block without a liberty is illegal.

A.4 Repetition

A whole-board situation is defined by:

- the colouring of the grid,
- the position of a possible prohibited intersection due to basic ko,
- the number of consecutive passes, and
- the player to move.

这个禁止重复的规则感觉跟SSK 有点不同

A repetition is a whole-board situation which is identical to a whole-board situation that occurred earlier in the game.

而且这里重复后的胜负计算更像是日本规则

If a repetition occurs, the game ends and is scored directly based on an analysis of all moves in the cycle starting from the first occurrence of the whole-board situation. If the number of pass moves in the cycle is identical for both players the game ends as a draw. Otherwise, the game is won by the player that played the most pass moves in the cycle. Optional numeric scores for an exceptional end by repetition are: $+\infty$ (black win), $-\infty$ (white win), and 0 (draw).

A.5 End

Normally, the game ends by 2 consecutive passes.[1] Only if a position contains a prohibited intersection due to basic ko, and the previous position did not contain a prohibited intersection due to basic ko, the game ends by 3 consecutive passes.

When the game ends it must be scored.

[1] Alternative: the game always ends after two consecutive passes.

A.6 Definitions for scoring

One-sided play consists of moves that may play stones on the board for one colour only (the other player always passes). Alternating play consists of alternating moves of both players.

Stones that cannot be captured under one-sided play of the opponent are unconditionally alive.

A region is an arbitrary set of connected intersections regardless of colour. The border of a region consists of all intersections which are not part of, but are adjacent to intersections of that region. A local region is a region which

- contains no unconditionally alive stones, and
- has no border or has a border that is completely occupied by unconditionally alive stones.

Each local region is fixed based on the arrangement of stones on the grid at the start of the scoring phase. Play is local if all moves are played in one local region, passes are also allowed.

A connector is an intersection of a region which has at least two adjacent intersections that are part of that region. A unique connector is a connector that would split the region into two or more disconnected regions if it would be removed from the region.

A single-point eye is an empty intersection which is

- not a unique connector, and
- not adjacent to any other empty intersection, and
- adjacent to stones of only one colour.

Blocks of stones that cannot become adjacent to at least one single-point eye under local one-sided play of their colour are dead. Blocks of stones that can become adjacent to at most one single-point eye under local one-sided play of their colour, and can be captured under local alternating play with their colour moving first are dead.[2]

A.7 Scoring

First, dead stones are removed from the grid. Second, empty blocks that are adjacent to stones of one colour only, get the colouring of those adjacent stones.[3] Finally, the score is determined by the number of black intersections minus the number of white intersections (plus komi). If the score is positive Black wins the game, if the score is negative White wins the game, otherwise the game is drawn.

[2]Optional addition: any block of stones with one liberty is dead.

[3]Optional addition: Third, if an empty block is adjacent to stones of more than one colour each empty intersection which is closer to a black stone becomes black and each empty intersection which is closer to a white stone becomes white. (If the distance is equal the colour remains empty.)

Summary

The thesis describes our research results and the development of new AI techniques that improve the strength of Go programs. In chapter 1, we provide some general background information and introduce the topics of the thesis. We focus on two important lines of research that have proved their value in domains which are related to and relevant for the domain of computer Go. These lines are: (1) searching techniques, which have been successful in games such as Chess, and (2) learning techniques, which have been successful in other games such as Backgammon, and in other complex domains such as image recognition. For both lines we investigate the question to what extent these techniques can be used in computer Go.

Chapter 2 introduces the reader to the game of Go. Go is the most complex popular board game in the class of two-player zero-sum perfect-information games. It is played regularly by millions of players in many countries around the world. Despite several decades of AI research, and a million-dollar prize for the first computer program to defeat a professional Go player, there are still no Go programs that can challenge a strong human player.

Chapter 3 introduces the standard searching techniques used in this thesis. We discuss minimax, $\alpha\beta$, pruning, move ordering, iterative deepening, transposition tables, enhanced transposition cut-offs, null windows, and principal variation search. The searching techniques are applied in two domains with a reduced complexity to be discussed in chapters 4 and 5.

Chapter 4 investigates searching techniques for the task of solving the capture game, a simplified version of Go aimed at capturing stones, on small boards. The main result is that our program PONNUKI solved the capture game on empty square boards up to size 5×5 (a win for the first player in 19 plies). The 6×6 board is solved, too (a win for the first player in 31 plies), under the assumption that the first four moves are played in the centre. These results were obtained by a combination of standard searching techniques, some standard enhancements adapted to exploit domain-specific properties of the game, and a novel evaluation function. We conclude that standard searching techniques and enhancements can be applied effectively for the capture game, especially when they are restricted to small regions of fewer than 30 empty intersections. Moreover, we conclude that our evaluation function performs adequately at least for the task of capturing stones.

Chapter 5 extends the scope of our searching techniques to Go, and applies them to solve the game on small boards. We describe our program MIGOS in detail. It uses principal variation search with (1) a combination of standard search enhancements (transposition tables, enhanced transposition cut-offs), improved search enhancements (history heuristic, killer moves, sibling promotion), and new search enhancements (internal unconditional bounds, symmetry lookups), (2) a dedicated heuristic evaluation function, and (3) a method for static recognition of unconditional territory. In 2002, MIGOS was the first Go program in the world to have solved Go on the 5×5 board.

We analyse the application of the situational-super-ko rule (SSK), identifying several problems that can occur when the history of a position is discarded by the transposition table, and investigate some possible solutions. Most problems can be overcome by only allowing transpositions that are found at the same depth in the search tree. The remaining problems are quite rare, especially in combination with a decent move ordering, and can often be ignored safely.

We conclude that, on current hardware, provably correct solutions can be obtained within a reasonable time frame for confined regions of a size up to about 28 intersections. For efficiency of the search, provably correct domain-specific knowledge is essential to obtain tight bounds on the score early in the search tree. Without such domain-specific knowledge, detecting final positions by search alone becomes unreasonably expensive.

Chapter 6 introduces the learning techniques used in this thesis. We explain the purpose of learning, and give a brief overview of the learning techniques that can be used for game-playing programs. Our focus is on multi-layer perceptron (MLP) networks. We discuss the strengths and weaknesses of the possible network architectures, representations and learning paradigms, and test our ideas on the simplified domain of connectedness between stones. We find that training and applying complex recurrent network architectures with reinforcement learning is quite slow, and that simpler network architectures may provide a good alternative especially when large amounts of training examples and well-chosen representations are available. Consequently, in the following chapters we focus on supervised learning techniques for training simple architectures to evaluate moves and positions.

Chapter 7 presents techniques for learning to predict strong moves from game records. We introduce a training algorithm which is more efficient than standard fixed-target implementations due to the avoidance of needless weight adaptation when predictions are correct. As an extra bonus, the algorithm reduces the number of gradient calculations as the performance grows, thus speeding up the training. A major contribution to the performance is the use of feature-extraction methods. Feature extraction reduces the training time while increasing the quality of the predictor. Together with a sensible scaling of the original features and an optional second-phase training, superior performance over direct-training schemes can be obtained.

The predictor can be used for move ordering and forward pruning in a full-board search. The performance obtained on ranking professional moves indicates that a large fraction of the legal moves may be pruned directly. Ex-

periments with the program GNU Go indicate that a relatively small set of high-ranked moves is sufficient to play a strong game against other programs.

Our conclusion is that it is possible to train a learning system to predict good moves most of the time with a performance at least comparable to strong kyu-level players. Such a performance can be obtained from a simple set of locally computable features, thus ignoring a significant amount of information which can be obtained by more extensive (full-board) analysis or by specific goal-directed searches. Consequently, there is still significant room for improving the performance further.

Chapter 8 presents learning techniques for scoring final positions. We describe a cascaded scoring architecture (CSA*) that learns to score final positions from labelled examples, and apply it to create a reliable collection of 9×9 game records (which is re-used for the experiments in chapters 9 and 10). On unseen game records CSA* scores around 98.9% of the positions correctly without any human intervention. By comparing numeric scores and counting unsettled interior points nearly all incorrectly scored final positions can be detected (for verification by a human operator). Although some final positions are assessed incorrectly by CSA*, it turns out that many are in fact scored incorrectly by the players. Detecting games that were incorrectly scored by the players is important for obtaining reliable training data.

We conclude that for the task of scoring final positions supervised learning techniques can provide a performance at least comparable to reasonably strong kyu-level players. This performance is obtained by a cascade of relatively simple classifiers in combination with a well-chosen representation, which only employs features that are calculated statically (without search).

Chapter 9 focuses on predicting life and death. The techniques from chapter 8 are extended to train MLP classifiers to predict life and death in non-final positions too. Experiments show that, averaged over the whole game, around 88% of all blocks (that are relevant for scoring) are classified correctly. Ten moves before the end of the game 95% of all blocks are classified correctly, and for final positions over 99% are classified correctly. We conclude that supervised learning techniques in combination with a well-chosen representation can be applied quite well for the task of predicting life and death in non-final positions. At least for positions near the end of the game we are confident that the performance is comparable to that of reasonably strong kyu-level players.

Chapter 10 investigates various learning techniques for estimating potential territory. Several direct and trainable methods for estimating potential territory are discussed. We test the performance of the direct methods, known from the literature, which do not require an explicit notion of life and death. Additionally, two enhancements for adding knowledge of life and death and an extension of Bouzy's method are presented. The experiments show that without explicit knowledge of life and death the best direct method is Bouzy's method extended with a means to divide the remaining empty intersections. When information about life and death is used to remove dead stones, the difference with distance-based control and influence-based control becomes small, and it is seen that all three methods perform quite well.

The trainable methods are new and can be used as an extension of the system for predicting life and death presented in chapter 9. A simple representation for our trainable methods suffices to estimate potential territory at a level outperforming the best direct methods. Experiments show that all methods are greatly improved by adding knowledge of life and death, which leads us to conclude that good predictions of life and death are the most important ingredient for an adequate full-board evaluation function.

Here we conclude that supervised learning techniques can be applied quite well for the task of estimating potential territory. When provided with sufficient training examples, these techniques easily outperform the best direct methods known from literature. On a human scale, we are confident that for positions near the end of the game the performance is at least comparable to that of reasonably strong kyu-level players. However, without additional experiments it is difficult to say whether the performance is similar in positions that are far away from the end of the game.

Finally, chapter 11 revisits the research questions, summarises the main conclusions, and provides directions for future research. Our conclusion is that domain-specific knowledge is the most important ingredient for improving the strength of Go programs. For small problems sufficient provably correct knowledge can be implemented by human experts. When searches are confined to regions of about 20 to 30 intersections, the current state-of-the-art searching techniques together with adequate domain-specific knowledge representations can provide strong and often even perfect play. Larger problems require heuristic knowledge. Although heuristic knowledge can be implemented by human experts, too, this tends to become quite difficult when the playing strength of the program increases. To overcome this problem learning techniques can be used to extract automatically and intelligently knowledge from the game records of human experts. For maximising performance, the main task of the programmer then becomes providing the learning algorithms with adequate representations and large amounts of reliable training data. When both are available the quality of the static knowledge that can be obtained using learning techniques is at least comparable to that of reasonably strong kyu-level players. The various techniques presented in this thesis have been implemented in the Go program MAGOG which enabled it to win the bronze medal in the 9×9 Go tournament of the 2004 Computer Olympiad in Ramat-Gan, Israel.

Samenvatting

Dit proefschrift beschrijft onderzoek naar en de ontwikkeling van nieuwe AI-technieken om de speelsterkte van Go-programma's te verbeteren. Hoofdstuk 1 geeft enige algemene achtergrondinformatie en introduceert de onderwerpen van dit proefschrift. We richten ons op twee belangrijke onderzoekslijnen die hun waarde hebben bewezen in domeinen die verwant zijn met het domein van computer-Go. Het zijn: (1) zoektechnieken, die succesvol zijn geweest in spelen zoals schaken, en (2) leertechnieken, die succesvol zijn geweest in andere spelen zoals Backgammon, en in andere complexe domeinen zoals beeldherkenning. Voor beide onderzoekslijnen beschouwen we de vraag in hoeverre deze technieken bruikbaar zijn in computer-Go.

Hoofdstuk 2 introduceert het spel Go. Go is het meest complexe populaire bordspel in de klasse van tweepersoons nulsom spelen met volledige informatie. Wereldwijd zijn er miljoenen mensen die regelmatig Go spelen. Ondanks tientallen jaren van AI-onderzoek, en een prijs van zo'n 1 miljoen dollar voor het eerste programma dat een professionele speler zou verslaan, maken Go-programma's nog altijd geen schijn van kans tegen sterke amateurs.

Hoofdstuk 3 bevat een overzicht van de standaard zoektechnieken die in dit proefschrift worden gebruikt. We behandelen minimax, $\alpha\beta$, snoeien, het ordenen van zetten, iteratief verdiepen, transpositie-tabellen, *enhanced transposition cut-offs*, *null windows*, en *principal variation search*. De zoektechnieken worden toegepast in twee domeinen met een gereduceerde complexiteit. Zij worden behandeld in de hoofdstukken 4 en 5.

Hoofdstuk 4 richt zich op zoektechnieken voor het oplossen van Slag-Go (een vereenvoudigde versie van Go gericht op het slaan van stenen) op kleine borden. Het belangrijkste resultaat is dat ons programma Ponnuki Slag-Go heeft opgelost voor kleine lege borden tot de grootte van 5×5 (de eerste speler wint na 19 zetten). Het 6×6 bord is ook opgelost onder de aanname dat de eerste vier zetten in het centrum worden gespeeld (de eerste speler wint na 31 zetten). Deze resultaten zijn verkregen met behulp van (1) standaard zoektechnieken, (2) enkele standaard verbeteringen waarbij gebruik gemaakt is van domeinspecifieke eigenschappen van het spel, en (3) een nieuwe evaluatiefunctie. We concluderen dat standaard zoektechnieken en de verbeteringen effectief toegepast kunnen worden voor het spel Slag-Go, met name als het domein beperkt is tot kleine gebieden met minder dan 30 lege kruispunten. Verder concluderen we dat de evaluatiefunctie in ieder geval geschikt is voor het vangen van stenen.

In hoofdstuk 5 breiden wij het domein van de zoektechnieken uit van Slag-Go naar Go, en passen ze toe bij het oplossen van Go op kleine borden. We beschrijven het programma MIGOS in detail. Het maakt gebruik van *principal variation search* met een combinatie van standaard verbeteringen (transpositie-tabellen, *enhanced transposition cut-offs*), aangepaste verbeteringen (*history heuristic, killer moves, sibling promotion*), en nieuwe verbeteringen (*internal unconditional bounds, symmetry lookups*), een heuristische evaluatiefunctie, en een methode voor het statisch herkennen van onconditioneel gebied. In 2002 heeft MIGOS als eerste programma ter wereld Go op het 5×5 bord opgelost.

Voorts analyseren we in dit hoofdstuk de toepassing van de situationele-super-ko regel (SSK). We beschrijven verscheidene problemen die zich kunnen voordoen in combinatie met de transpositie-tabel als de geschiedenis van een positie wordt genegeerd, en onderzoeken enkele mogelijke oplossingen. De meeste problemen zijn oplosbaar door alleen transposities te gebruiken die op dezelfde diepte in de zoekboom gevonden zijn. De resterende problemen zijn zeldzaam, vooral in combinatie met een degelijke zettenordening, en kunnen meestal veilig genegeerd worden.

We concluderen dat, op hedendaagse hardware, bewijsbaar correcte oplossingen verkregen kunnen worden binnen een redelijke tijd voor afgesloten gebieden met maximaal zo'n 28 kruispunten. Voor efficiënt zoeken is bewijsbaar correcte domeinspecifieke kennis essentieel voor het verkrijgen van stringente grenzen aan de mogelijke scores in de zoekboom. Zonder gebruik te maken van zulke domeinspecifieke kennis is het herkennen van eindposities, puur op basis van zoeken, in de praktijk niet goed mogelijk.

Hoofdstuk 6 geeft een overzicht van de leertechnieken die in dit proefschrift gebruikt worden. We leggen het doel van het leren uit, en geven aan welke leertechnieken gebruikt kunnen worden. Onze aandacht is gericht op meerlaags perceptron (MLP) netwerken. We bespreken de sterke en zwakke kanten van mogelijke architecturen, representaties en leerparadigma's, en testen onze ideeën op het vereenvoudigde domein van connectiviteit tussen stenen. Onze bevindingen laten zien dat het trainen en toepassen van complexe recurrente netwerk-architecturen met *reinforcement learning* bijzonder traag is, en dat eenvoudiger architecturen een goed alternatief zijn, vooral als grote aantallen leervoorbeelden en goed gekozen representaties beschikbaar zijn. Derhalve richten wij ons in de hoofdstukken 7, 8, 9 en 10 op *supervised* leertechnieken voor het trainen van eenvoudige architecturen voor het evalueren van zetten en posities.

Hoofdstuk 7 presenteert technieken voor het leren voorspellen van sterke zetten op basis van opgeslagen partijen. We introduceren een trainingsalgoritme dat efficiënter is dan standaard implementaties omdat het geen onnodige berekeningen uitvoert wanneer de ordeningen correct zijn en daarmee het aantal noodzakelijke gradiënt-berekeningen reduceert, wat de training aanzienlijk versnelt naarmate de voorspellingen beter worden. Een belangrijke bijdrage aan de prestatie is het gebruik van kenmerk-extractie. Kenmerk-extractie reduceert de leertijd en verbetert de kwaliteit van de voorspellingen. In combinatie met een zinnige schaling van de originele kenmerken en een tweede-fase training kunnen superieure prestaties worden verkregen ten opzichte van directe training.

De voorspellingen kunnen gebruikt worden om zetten te ordenen en voorwaarts te snoeien bij het zoeken op het volledige bord. De kwaliteit van de ordening van professionele zetten geeft aan dat een groot deel van de legale zetten direct gesnoeid kan worden. Experimenten met het programma GNU Go laten zien dat een relatief klein aantal hoog geordende zetten voldoende is om een sterke partij te spelen tegen andere programma's.

Onze conclusie is dat het mogelijk is om een lerend systeem te trainen dat zetten kan voorspellen op een niveau dat minstens vergelijkbaar is met dat van sterke kyu-spelers. Dit niveau is reeds haalbaar op basis van relatief eenvoudige lokaal te berekenen kenmerken, waarmee we dus een aanzienlijke hoeveelheid informatie negeren die verkregen kan worden door middel van een uitgebreidere analyse (van het volledige bord) of door middel van het speciaal doelgericht zoeken. Derhalve zijn er nog voldoende verbeteringen mogelijk.

Hoofdstuk 8 presenteert leertechnieken voor het waarderen van eindposities. We beschrijven een cascade-scoring architecture (CSA*) die leert om eindposities te waarderen op basis van gelabelde voorbeelden. We passen deze methode toe om een betrouwbare verzameling 9×9 partijen samen te stellen (die tevens gebruikt wordt voor de experimenten in hoofdstuk 9 en 10). Op onafhankelijke partijen scoort CSA* volledig zelfstandig zo'n 98,9% van de posities correct. Door numerieke scores te vergelijken en onbesliste interne punten te tellen kunnen bijna alle incorrect gescoorde eindposities gedetecteerd worden (voor verificatie door een menselijke operator). Hoewel sommige eindposities incorrect beoordeeld worden door CSA*, blijkt dat het merendeel incorrect beoordeeld is door de spelers. Het herkennen van partijen die incorrect beoordeeld zijn door de spelers is belangrijk voor het verkrijgen van betrouwbaar leermateriaal.

We concluderen dat *supervised* leertechnieken voor het waarderen van eindposities een prestatieniveau halen dat zeker vergelijkbaar is met redelijk sterke kyu-spelers. Dit niveau wordt verkregen met behulp van relatief eenvoudige beslissers in combinatie met een goed gekozen representatie, die slechts gebruik maakt van kenmerken die statisch berekend kunnen worden (zonder zoeken).

Hoofdstuk 9 richt zich op het voorspellen van leven en dood. De technieken uit hoofdstuk 8 worden uitgebreid voor het trainen van MLP-beslissers om leven en dood te voorspellen in niet-eindposities, ook op basis van gelabelde voorbeelden. De experimenten laten zien dat, gemiddeld over de gehele partij, zo'n 88% van alle blokken (die relevant zijn voor de score) correct geclassificeerd worden. Tien zetten voor het einde van de partij wordt zo'n 95% van alle blokken correct geclassificeerd, en voor de eindposities wordt meer dan 99% correct geclassificeerd. We concluderen dat *supervised* leertechnieken in combinatie met een adequate representatie behoorlijk goed in staat zijn om ook in niet-eindposities leven en dood te voorspellen. We zijn er van overtuigd dat, in ieder geval voor posities vlak voor het einde van de partij, het prestatieniveau vergelijkbaar is met dat van redelijk sterke kyu-spelers.

Hoofdstuk 10 onderzoekt verscheidene leertechnieken voor het schatten van potentieel gebied. Verschillende directe en lerende methoden voor het schatten van potentieel gebied worden behandeld. We testen de prestaties van de directe methoden die bekend zijn uit de literatuur en geen expliciete notie van leven

en dood nodig hebben. Tevens presenteren we twee verbeteringen voor het toevoegen van kennis omtrent leven en dood en een uitbreiding van Bouzy's methode. De experimenten laten zien dat zonder expliciete kennis van leven en dood Bouzy's methode, indien die uitgebreid is met een methode om de neutrale punten verder te verdelen, de beste directe methode is. Als informatie over leven en dood gebruikt wordt om de dode stenen te verwijderen, is het verschil met *distance-based control* en *influence-based control* klein, en doen alle drie de methoden het behoorlijk goed.

De lerende methoden zijn nieuw en kunnen gebruikt worden als uitbreiding van het systeem voor het voorspellen van leven en dood dat we beschreven hebben in hoofdstuk 9. Een eenvoudige representatie voldoet om potentieel gebied beter te schatten dan de beste directe methoden. Experimenten laten zien dat alle methoden aanzienlijk beter presteren met kennis van leven en dood. Dit leidt tot de conclusie dat kennis van leven en dood het belangrijkste ingrediënt is van een adequate evaluatiefunctie voor het gehele bord.

Hier concluderen wij dat *supervised* leertechnieken behoorlijk goed toegepast kunnen worden voor de taak van het schatten van potentieel gebied. Wanneer voldoende trainingsvoorbeelden beschikbaar zijn kunnen deze technieken de beste directe methoden uit de literatuur eenvoudig verslaan. Op de menselijke schaal zijn we ervan overtuigd dat vlak voor het einde van de partij de prestaties minstens vergelijkbaar zijn met die van redelijk sterke kyu-spelers. Echter, zonder aanvullende experimenten is het lastig om te zeggen of dat ver voor het einde ook zo is.

Tenslotte komen we in hoofdstuk 11 terug op de onderzoeksvragen, geven een overzicht van de belangrijkste conclusies, en eindigen met suggesties voor nader onderzoek. Onze conclusie is dat domeinspecifieke kennis het belangrijkste ingrediënt is voor het verbeteren van de sterkte van Go programma's. Voor kleine problemen kan voldoende bewijsbaar correcte kennis geïmplementeerd worden door menselijke experts. Als het zoeken begrensd is tot gebieden van ongeveer 20 tot 30 kruispunten zijn de huidige *state-of-the-art* zoektechnieken in combinatie met adequate domeinspecifieke kennisrepresentaties in staat tot zeer sterk en vaak zelfs perfect spel. Voor het aanpakken van grotere problemen is heuristische kennis noodzakelijk. Hoewel heuristische kennis ook geïmplementeerd kan worden door menselijke experts wordt dit naarmate het programma sterker wordt snel lastiger. Dit laatste kan vermeden worden door gebruik te maken van leertechnieken die automatisch en op intelligente wijze kennis extraheren uit de partijen van menselijke experts. Voor het maximaliseren van de prestaties wordt de belangrijkste taak van de programmeur dan het leveren van adequate representaties en grote hoeveelheden betrouwbare leervoorbeelden aan het leeralgoritme. Als beide beschikbaar zijn is de kwaliteit van de statische kennis die kan worden verkregen met leertechnieken minstens vergelijkbaar met die van spelers van redelijk sterk kyu-niveau. De technieken die we in dit proefschrift hebben gepresenteerd zijn geïmplementeerd in het Go-programma MAGOG en hebben direct bijgedragen aan het winnen van de bronzen medaille in het 9×9 Go-toernooi van de 9de Computer Olympiade (2004) in Ramat-Gan, Israel.

Curriculum Vitae

Erik van der Werf was born in Rheden, The Netherlands, on May 30, 1975. From 1987 to 1993 he attended secondary school (Atheneum) at the Thomas a Kempis College in Arnhem. From 1993 to 1999 he studied Applied Physics at the Delft University of Technology (TU Delft). During his years in Delft he was an active member of the student association D.S.V. Sint Jansbrug, and for a while he also studied psychology at the Universiteit Leiden. In 1998 he did a traineeship at Robert Bosch GMBH in Hildesheim, Germany. There he did research on neural networks, image processing, and pattern recognition for automatic license plate recognition. In 1999 he received his M.Sc. degree, and the Dutch title of ingenieur (ir), by carrying out research in the Pattern Recognition Group on non-linear feature extraction by diabolo networks. In 2000 he started to work as a research assistant (AIO) at the Universiteit Maastricht. At the Institute of Knowledge and Agent Technology (IKAT), under auspices of the research school SIKS, he investigated AI techniques for the game of Go under the supervision of Jaap van den Herik and Jos Uiterwijk. The research resulted in several publications, this thesis, the first Go program in the world to have solved 5×5 Go, and a bronze medal at the 2004 Computer Olympiad. Besides performing scientific tasks, he was teaching (Machine Learning, Java, Databases), administrator for the Linux systems, a member of the University Council (the highest democratic body of the university, with 18 elected members), a member of the commission OOI (education, research, internationalisation), and an active Go player in Heerlen, Maastricht, and in the Rijn-Maas liga.

SIKS Dissertation Series

1998

1 Johan van den Akker (CWI[4]) *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*

[4]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; 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*