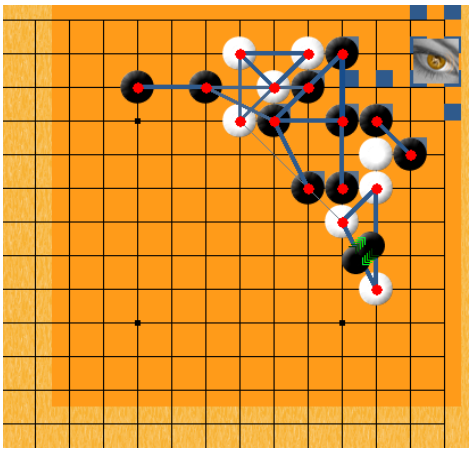
Learning Patterns in the Game of Go



MSc Thesis (*Afstudeerscriptie*)

written by

Emil H.J. Nijhuis
(born 22.5.1980 in Kiel)

under the supervision of **dr. Nikos Vlassis**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Artificial Intelligence

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
15.12.2006	dr. Nikos Vlassis
	dr. Peter van Emde Boas
	dr. ir. Leo Dorst

Abstract

This thesis introduces concepts for learning expert human positional classifiers in the game of Go. While most studies in the field of Go focus on learning methods to find the best move, this thesis focuses on learning methods for evaluating the quality of Go stones in a position. A data set of 2,638 Go Tesuji problems has been created. For these problems expert human classifiers have been recorded using a first order logic annotation extension developed for the current standardized format SGF (Smart Go File). The learning of the human classifiers has been conducted with Support Vector Machines using low level features such as the Relative Subgraph Features (RSF) from the Common Fate Graph (CFG). Relative Subgraph Path Features (RSPF) have been developed for learning connectivity and proved to be an appropriate representation for the learning task.

Acknowledgements

Making this thesis was the result of intensive study of two disciplines: Artificial Intelligence and Go. It is my experience that it is not easy to truly master a discipline without great help and proper guidance and that it is very difficult to excel in more than one discipline.

Throughout my life I was blessed with great teachers who taught, supported and believed in me. Without their wisdom and selfless efforts this thesis would not have become what it is now. So I would like to express my gratitude towards them for their help in my life. Most notably from all the professors who taught me during my studies at the University of Amsterdam, I would like to thank Nikos Vlassis for the supervision, inspiration and patience on this thesis. Also I would like to thank the Go players who taught me the game by just playing a game of Go or simply analyzing one. With special thanks to my brother Caspar, Hans Pietsch, Chizu Kobayashi and Guo Juan who tirelessly taught me throughout my Go career.

This thesis also has been reviewed and greatly improved by Jahn-Takeshi Saito, for which I am very thankful as well.

Finally I would like dedicate this thesis to my fiancée Yowon Choi, for her trust and devotion.

Preface

Reading this thesis requires some basic knowledge of the Game of Go. Readers unfamiliar with the game are therefore advised to start reading chapter 2 first, to get an impression of the game.

Writing a thesis is very difficult as it is. Reading a thesis is probably more difficult though, especially for a person who has not been involved in the creation of it. Reading a considerable amount of papers involving computer science and Go, I could not help but wonder whether a regular Go player with no scientific background would be able to read them. Equally the question arose whether researchers with no Go background would feel comfortable reading them. I tried to make it my merit to write this thesis in a way that it would be attractive to read no matter what background one has. I hope you will find yourself having as much pleasure reading it as I had writing it.

Table of Contents

Chapter 1

Introduction

For over 20 years Go has attracted computer scientists and programmers as an appealing research field. Many articles have been written covering various aspects of the game; also Go programs have been written, however, none of them has yet reached the level of an Amateur 1-dan, which is considered to be the entry level for an expert/master player. Remarkably, since the introduction of pattern recognition methods in Mark Boon's Goliath program [1] no significant improvements in the development of new Go programs have been made, while computer processing power has steadily increased. Since then many new techniques have been developed and applied [3, 4, 8, 11, 21] however none of these approaches resulted in a much needed boost in this research area. Perhaps one of the main reasons why no significant improvements have been made is the lack of digitally available expert knowledge. For a computer scientist without any expert/master level in Go it seems almost impossible to create a reasonable Go program without any well documented expert data. As a consequence the research area is almost restricted to Go experts with computer science background. In addition a programmer/researcher in this field has to spend a great deal of time to start up his project by creating a framework for displaying moves and extracting simple features (although to some extend frameworks are available online [2, 9, 12]).

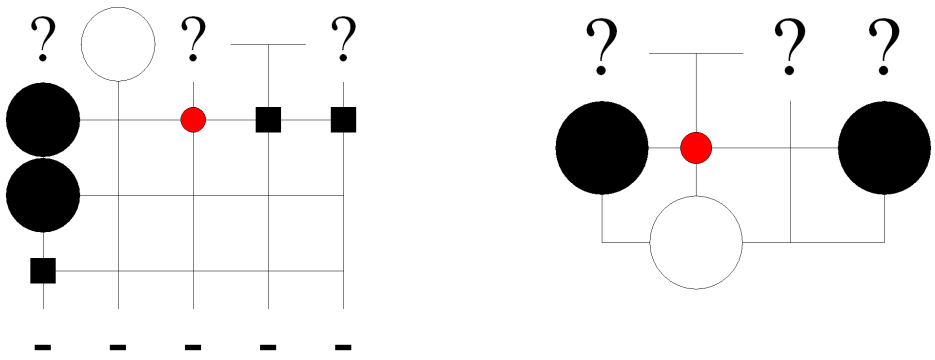
Altogether, these aspects make the research field of applied computer science to Go attractive because of the challenges involved, but at the same time there is a burden for interested researchers because of the commitment required to complete a research project, risking that the task ahead possibly will not deliver any significant improvement or clear results to current research. For *"life and death"* problems, a crucial part of the game, several techniques have been developed capable of delivering good move predictions [19, 31, 20]. But are there more of this

type of areas inside the Go domain? The main question of this thesis therefore deals with whether there are ways to make this research area more attractive and accessible for interested researchers and how experts can be involved in this process.

Jan Roman and Hendrik Blockeel presented in [23] some directions of possible research in the field of computer Go. Next to improving min-max search methods using temperature functions, they also suggest that it would be fruitful to learn high-level concepts such as secure territory and influence, as then decision processes for making moves can be more accurate. They also stress that the incremental building of such high-level concepts is one of the most important aspect for further research in the application of inductive logic programming to Go. As this field is relatively unexplored a good foundation on how to learn high-level concepts needs to be made. But moreover to even think about learning high-level concepts such as influence or secure territory we must have a reliable dataset were these concepts are included. So far the only electronically available expert data on Go are:

1. Records of expert players
2. Collections of *life and death* problems
3. Handmade pattern databases like the ones used in GnuGo [12]. This databases contain collections of abstract patterns which define what kind of move should be played to accomplish an outcome, see figure 1.1 for example patterns
4. *Joseki* (an optimal corner sequence) databases in which correct opening sequences are stored

None of these expert data reveal high-level concepts in a obvious way. Without expert knowledge of the game the most a researcher can see from those records is what the likely best next move in a given situation could be. Therefore there is a need for datasets with high-level concepts labeled by an expert. But more importantly a standard for recording high-level expert labels needs to be created to record them. For this reason was one of the main aspects of this thesis to create a recording standard and the collection of expert data. To prove the dataset's usability and also to show what kind of progress can be forecast by putting more resources in this type of interdisciplinary research, several tests have been conducted.



(a) A 5x5 pattern of the category *"edge block/expand"*

(b) A 4x3 pattern of the category *"cut/connection"*

Figure 1.1: Example patterns from GnuGo. Explanation: the red dot marks Black's next move, ■ marks a square either containing black or empty but not white. The symbol '-' marks the edge of the board indicating the patterns relevance is for edge situations. Finally '?' indicates a location where it does not matter what is there, except that it cannot be off the edge of the board.

Once several accurate high-level predictors have been developed for evaluating a position, they can be used to find a good next move by using temporal difference learning, a method first described by Rich Sutton in 1988 [28]. In backgammon where the evaluation of the quality of a position is easier, temporal difference learning has already been applied successfully. TD-gammon, the backgammon program written by Tesauro [29] is not only a great example of how a program can reach expert level in a highly complex game with machine learning, but is arguably the most impressive result so far in the field of applied learning methods. Also in Go experiments with temporal difference learning have been executed, inspired by the success of Tesauro. Schraudolph et al. [25] tried a similar approach, but with limited success compared to TD-gammon.

Of course temporal difference learning with high-level concept predictors would still be a challenge, but should be more promising than trying to learn directly from a board situation. Markus Enzenberger developed the Go program NeuroGo for 9x9 boards [10], which basically learns by connecting the Go board intersections to a neural network with several hidden layers for learning high-level concepts with considerable success. Graepel et al. [13] point out that all known approaches render the learning task too abstract, as the representations of board positions and candidate moves are too confined.

1.1 Learning Go

Before we can decide which methods we want to use for learning Go, it is important to outline what the learning task actually is. Or in other words, we need to outline what is important in Go and how we can learn these aspects.

Because of the simplicity of the Go rules a novice can learn the essence of the game within fifteen minutes. But just understanding these rules quickly leads to a couple of deceiving approaches to learn the game. As researchers in the field of Artificial Intelligence we are mainly interested in the methods used to solve our problems rather than the problems we are trying to solve. This creates some misconceptions on learning Go using AI methods, as essential aspects of the game are often overlooked and omitted. One of the main misconception is that the bottleneck of current software is the lack of reliable territory evaluation functions. Learning such a function (see [26] for an approach) in combination with an exhaustive min-max research would lead presumably to a breakthrough and bring Go programs to the next level. This view has been brought forward as the game is obviously decided by who has more territory at the end of the game. My experience with the game, as an Amateur 6-Dan, however brings me to the conclusion that this view indeed captures an important aspect of the game, but lacks the understanding of why humans can play this game at a so much higher level than current programs.

For several years I have been teaching Go to various people from novice to master level. It is my experience that the strength of a Go player does not lie in whether he or she is able to decide what move is bigger by looking at the distributions of the territories, but the skill of a player lies in the ability to fight simple or complex battles by being aware of their outcome and consequences. In fact one of the first things among the things I teach to novices is that Go is a battle for territory. This means that understanding the value of territory is important but even more essential is to understand how the game is fought. So what makes us humans strong at learning these two aspects of the game?

The game has 361 empty intersections at the beginning and starts filling up during the course of the game. In order not to waste thinking time about all those possibilities a player develops concepts to evaluate a position. One of the most basic concepts is the construct of a group, its strength, its influence, its weakness. With the notion of these concepts a high-level understanding is developed which is then used to determine short and long term goals. Finally these goals result in move considerations which are then evaluated within the context of the underlying

goals. Learning concepts involves the recognition of patterns of either structures or other high-level concepts or shapes appearing on the Go board.

The pattern recognition process is therefore essential to the learning and understanding of concepts in the game. The better the understanding of concepts in the game the better decisions can be made when playing. If concepts can be learnt from patterns by humans then there should be also a way for computer programs. This thesis tries to show ways how concepts can be learnt from patterns.

Simple pattern databases have already been developed in connection with a set of decision rules and conclusions which are followed when a pattern occurs. The part of learning concepts is usually skipped and left at the hands of the programmer. It seems therefore interesting whether this process can be improved by directly learning concepts from patterns using concepts labeled by experts. So far, no research has been done specifically on the learning of basic concepts in the game of Go. This thesis shows how learning the basic concepts of connectivity, move type and best next move can be achieved.

Chapter 2

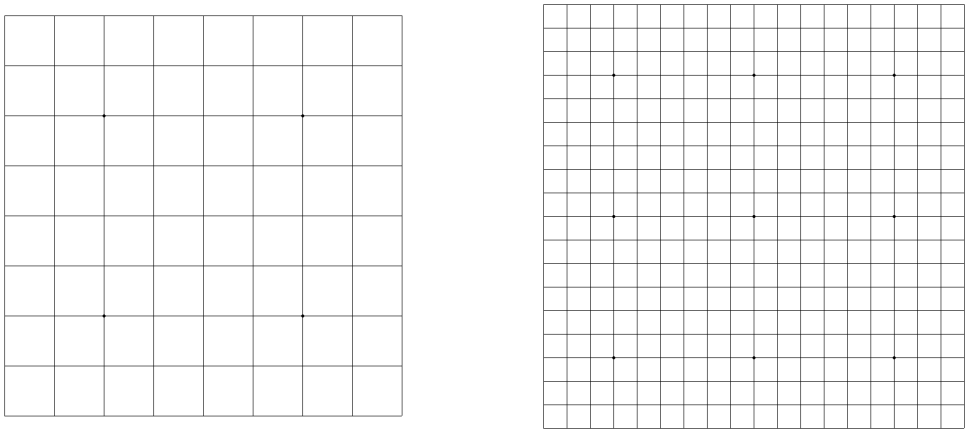
The game of Go

This chapter briefly describes the rules of the game. Also some of the concepts and notations used in this thesis are described in a separate section.

Go is a game played by two players with two sets of stones. One player takes the black stones and the other the white stones. The game is played on a board with 19 horizontal and vertical lines, see figure 2.1 (b). Beginners normally start playing the game on board with 9 horizontal and vertical lines to get accustomed to the rules of the game, see figure 2.1 (a). For the purpose of explaining the rules the 9x9 board is mostly used for illustrative reasons.

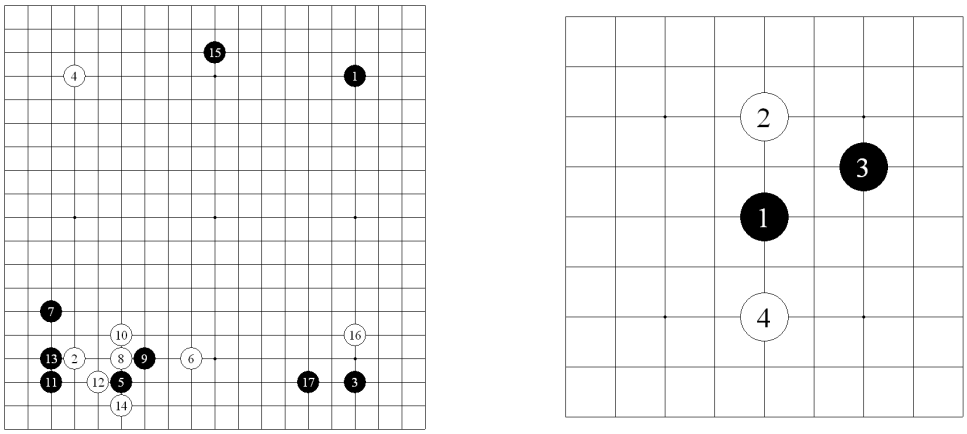
In the beginning the Go board is empty. The player with the black stones is allowed to play first. It is only allowed to play one stone at the time, which is placed on one of the intersections (which includes edge and corner intersections). Figure 2.2 shows a typical opening sequences.

A stone, once played, is not to be moved unless captured. If a stone or multiple stones of the same color are surrounded by the other color, such that no direct adjacent intersection is empty, then the stone(s) is (are) captured (see figure 2.3). Adjacent empty intersections are also referred to as *liberties*.



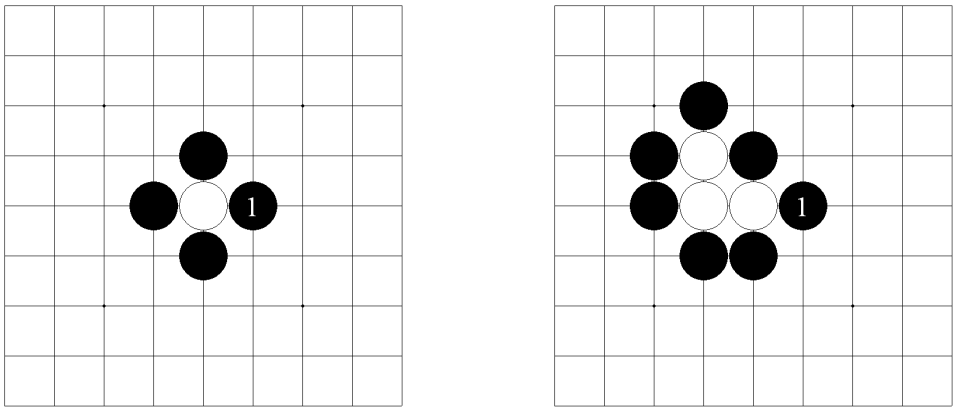
(a) 9x9 board (b) 19x19 board

Figure 2.1: Empty Go boards



(a) A 19x19 opening (b) A 9x9 opening

Figure 2.2: Opening sequences in the game of Go



(a) The white stone is removed after black played at 1.

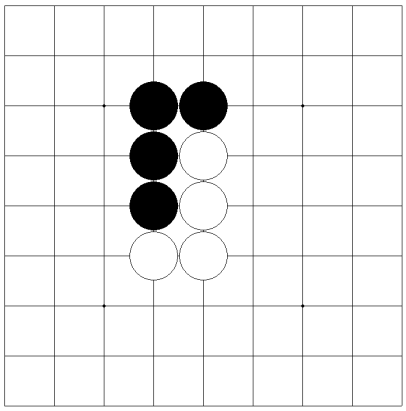
(b) The three white stones are removed after black played at 1.

Figure 2.3: The capturing rule

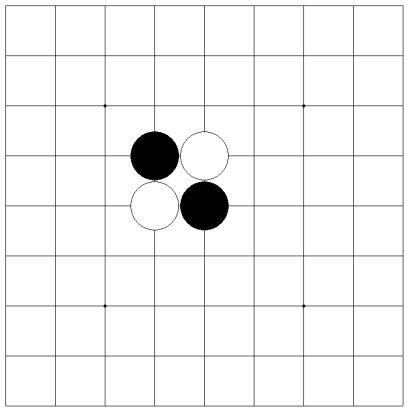
From the capturing rule for multiple stones it can be seen that stones are connected which are positioned on directly neighboring intersections. Diagonal neighboring intersections are not connected as there is no direct line between them, see figure 2.4. Connected stones share a common fate and cannot be captured separately. Stones which are connected are also called a *chain*. Note that a single stone is also a chain.

The goal of the game is to make territory. The player who has the largest territory at the end of the game is the winner. Territory can be empty surrounded intersections and captured enemy stones, see figure 2.5(a) and (b).

A situation where black and white could capture each other without interruption is called a *ko* (meaning indefinite in Japanese). Figure 2.6 (a) shows a typical ko situation. It is not allowed to recapture directly in a ko situation. If both players have played elsewhere before, it is allowed to recapture. The rule can be simplified by saying that it is not allowed to repeat a position on the Go board during a game.

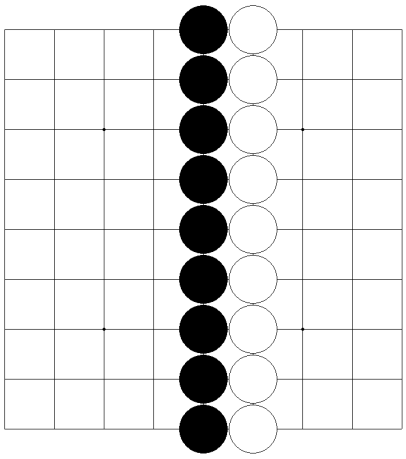


(a) The black stones are connected and so are the white ones.

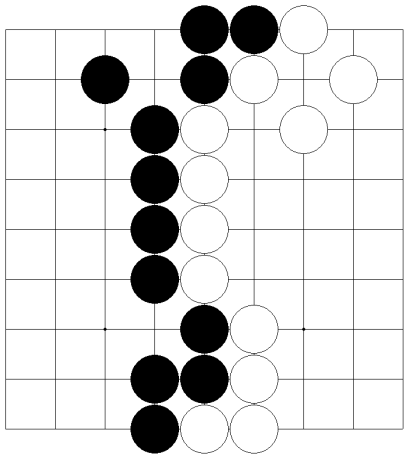


(b) The two black stones and the two white stones are separated.

Figure 2.4: The connectivity rule



(a) Black has 36 points and white has 27, hence Black wins by 9.



(b) White captured one black stone. So the score is 29 for Black and 28+1=29 for White.

Figure 2.5: The counting rule

The above explanations are sufficient for playing the game. There are some concepts and notations used inside the thesis which will be defined now, but are not part of the rule set.

2.1 Concepts and notations

In Go, advanced players label moves with a descriptive name. A diagonal move for example is called a *kosumi* or a one space jump is called an *ikken-tobi* or simply *tobi*. These names are referred to as *move types* inside this thesis. This is useful as moves can be labeled according to their property. Discussions between players about moves can than become easier as players can point out a move by its property instead of using coordinates. See figure 2.7 (a) for examples.

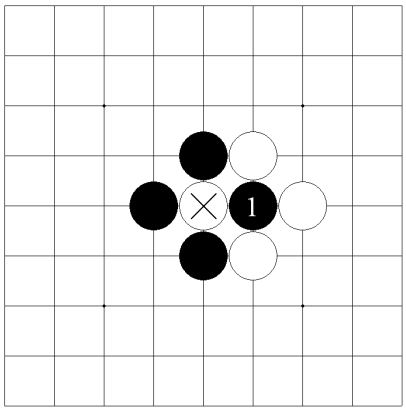
Another property of a move is the positional change it will inflict. A move which captures a stone is a capturing move. A move which threatens to capture a stone is called an *Atari*. A move which reduces territory is a reducing move and so on. This property of a move is referred to as *outcome type* in this thesis. See figure 2.7 (b) for examples.

Places which are important to both players are called vital points. In figure 2.7 (b) for instance, the white group at the left can survive if it would have been White's turn and he/she would play at the same spot as black. The place is therefore called a vital point.

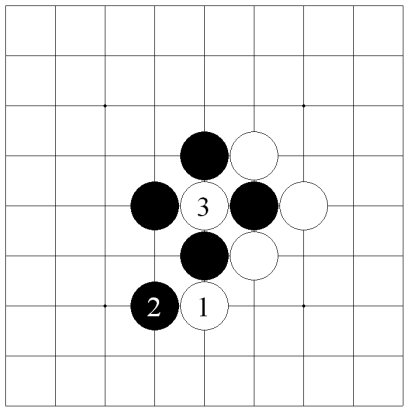
An concept used very often in Go is the one of a group. The term has already been used in the above contents. It identifies a set of chains which are related and have a common fate. A group is alive when it has two *eyes*. An eye consists of one empty or several empty connected intersection(s) where the surrounding chains can only be captured if a stone is placed in such a way that it does not have any liberties. Usually a stone is captured when it does not have any liberties left. If the last stone placed takes the last liberty of other stones then those stones will removed and the last placed stone remains on the Go Board. Therefore a group is alive if it has *two eyes*. See figure 2.8 for examples.

Stones which are not part of the same chain but cannot be separated because of the rules imposed by the game and optimal play are also called connected. In figure 2.7(a) are the black stones connected for instance, as Black has two possible moves to connect his stones to a single chain. If a chain A is connected with another chain B then this fact is abbreviated with the notation $A \longleftrightarrow B$.

The above explanations of the game are very concise and the reader who is unfamiliar with the game can visit an online lecture by Jan van der Steen for more examples at `http://gobase.org/studying/rules/`.

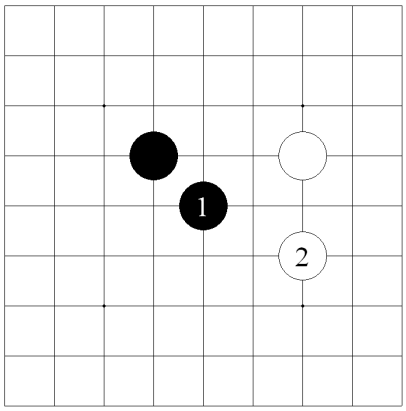


(a) Black has just captured the white stone marked with X. White is not allowed to recapture directly.

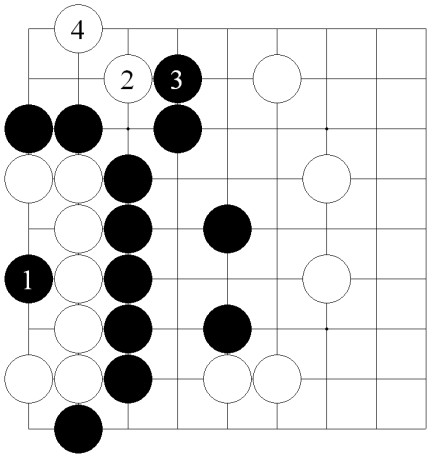


(b) White plays elsewhere first with 1 and Black responds 2, finally White can recapture with 3.

Figure 2.6: The ko rule

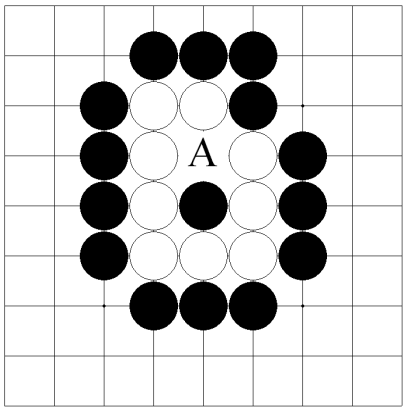


(a) Black plays a *kosumi* and White responds with an *ikken-tobi*.

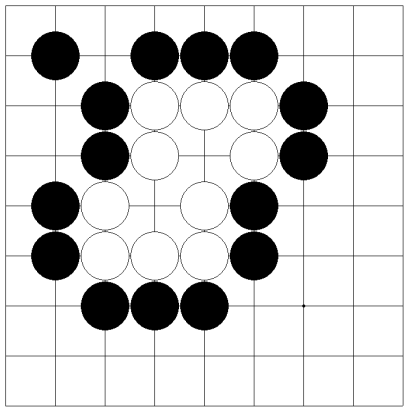


(b) Black kills the white stones at the left by playing the vital point. White replies by reducing Black's territory in the top left corner with a *peeping* move.

Figure 2.7: Move type, outcome type and vital point examples



(a) The white group has one eye. Black can capture the white group by playing at A. The black stone would then take its last liberty, but because it takes the last white liberty as well the white group can be captured.



(b) The white group has two eyes and cannot be captured. If Black plays into one of the eyes the stone would be automatically captured.

Figure 2.8: Example of eyes

Chapter 3

The XGF-standard

Today many AI-approaches in the field of Go research involve the use of professional or master level Go records for extracting knowledge. In David Stern's approach [27] move predictions are the result of learning a Bayesian pattern ranking from 181,000 expert games. The idea of this approach was inspired by Frank de Groots Go software Moyogo [14], which contains a SQL database of 450,000 games. The incorporation of this technique into Go playing software has also been tried, but the success and impact of this approach remains to be seen yet.

Although de Groot and Stern's success can be considered as remarkable, the question that arises here from a Go player's point of view is: why are hundreds of thousands of Go records not enough to learn the game at master level? The question might seem dubious to a Go programmer, as writing a playing Go program is extremely difficult, but if we put in perspective that professional Go players rarely replay over 5,000 games during their Insei (professional aspirant) education, then the value and effectiveness of the game records needs to be put into question. Also when we look at how humans learn the game, learning from expert games seems premature, as most players would study high-level games only after reaching a certain understanding of the game themselves. Expert Go players encourage studying expert games for two learning purposes. Firstly, it is important for a player to get a feeling of what correct and incorrect moves and good and bad shapes are. Secondly, people study expert games to understand the thought and evaluation processes of the players, e.g., when a situation is settled on the board and does not require any more attention with another move, or how strong a group is and what the appropriate distance to that group would be. Both these purposes usually require the player to have some basic knowledge of the game in the first place.

But as far as the Bayesian pattern ranking approach goes, it seems a good way to learn correct and incorrect moves based on shapes without the understanding of the local and global thinking and evaluation process.

So the question remains how can a Go program learn more from the available material? Are the learning methods used adequate enough to deal with a problem of this complexity? Is the learning material rich enough so that proper learning is possible or is it necessary to develop more detailed and descriptive material? It is hard to point the finger on where the learning process can be improved, but the answer to the problem probably involves a little bit of everything. While it might be fairly hard to create better methods for learning Go because it is a problem without a known solution, it is actually quite easy to create better learning material for computers. Easy in the sense that we know that we can attach more meaning to a Go record than only the move order involved. The simplest way to illustrate that is by asking an expert player to explain any Go position. Most likely the expert would come up with various responses, such as: "the black group in the upper left corner is safe and also puts pressure on the white group on the right", "the black group in middle is very vulnerable, but if it will survive or a part of it Black will win the game by points easily" or "White's territory does not look solid yet, Black should look for ways to exploit its weaknesses".

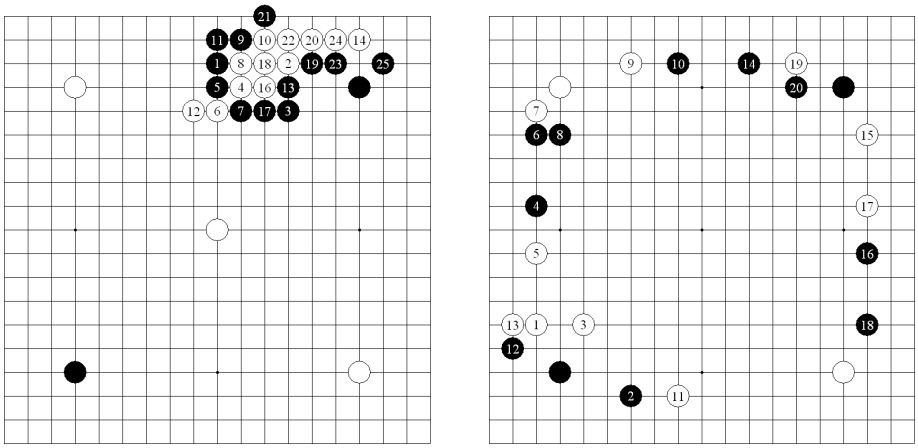
Of course we could also ask the expert where he or she believes the best next move would be, but without understanding the concepts of the game that might not be very useful to a learning person or program. As this labeling of attributes on the groups, territories, positions are hard to derive by just replaying a game record, it seems interesting to try a more subtle approach. The straightforward thing to do would be to record everything an expert player says on a position and keep his/her comments in a text file, however that would lead to complications of the file interpretation. Obviously it would be better to define certain attributes first and then let an expert player understand the attributes and subsequently let him or her explain positions using the attributes. But for precisely this reason a well elaborated standard for recording expert knowledge needs to be developed. In the next section a short introduction on current available Go documents will be given to show their significance in history, and in the final section a new standard will be proposed to improve the quality of today's learning material for computer Go programs.

3.1 A short introduction to Go documents

Searching for new ways to create learning material for Go programs, it seems a good idea to look back in history to see how Go has developed over the course of time. Go related documents played a crucial role in the study of Go as they preserved the knowledge of players who studied the game intensively. In the present time the quality of play of today's strongest players keeps improving and documents catalyze the process for upcoming generations to reach the highest level.

3.1.1 The history of Go documents

Although it is not known when Go exactly was invented most sources reveal that the game is about 4,000 years old. Since then the quality of how the game is played evolved to today's standards. Also the rules of the game have slightly changed, such as the starting position (ancient Chinese game records show that games were beginning with preset stones on the starpoints, see figure 3.1 for examples) and complex ko rules.



(a) Extracted Go record from Xuanxuan Qijing, 1347AD

(b) Wu Diagram, around 230AD

Figure 3.1: ancient Go records

The first discovered Go game record, called the "Wu Diagram", dates from around 230AD in China. Which means that before that time the game has been passed on from generation to generation by student-mentor relationships. The

quality of the Wu-Diagram also provides an indication that the master level of today's 1-dan may already have been reached by a player in that time. Of course since then many new strategy concepts and opening insights have been established, but by only observing the tactical finesse of those players it can be concluded that the overall skill of them would still be close to master level.

The Chinese Go book Xuanxuan Qijing from around 1350AD is a collection of ingenious Go problems, records and essays, which shows another milestone in the history of Go records. Its significance and quality has been proven timeless as many professional aspirants still use the Go problems to train their ability to read complex board situations. Without any doubt were the tactical skills of players from that time already very high.

But it was not until the beginning of the 17th century when Go study flourished in Japan because several study groups, the so called houses, Honinbo, Inoue, Hayashi, Yasui, were competing for the position of being the Shogun's personal teacher, the Godokoro. This not only resulted in a very competitive and fruitful environment for progress in the understanding of the game, but also in study material in the form of game records. Most notable are the records sometimes referred to as the Castle Games, where two players from opposing houses would fight for several days on a single game to be presented later on to the Shogun as a form of art and entertainment.

In 1924 the Nihon Ki-in, the Japanese Go Association, was founded, largely responsible for the writing of many of today's popular Go literature, such as commentary of professional games in magazines, *fuseki* (opening), *joseki* (corner sequences) and *tsumego* (life and death problems) encyclopedias, various study material covering a variety of topics ranging from beginner to master level. In the second half of the 20th century Go organizations in China, Korea but also in Taiwan followed suit and started their professional organizations of their own. Today Japan, Korea, China and Taiwan play a central role in the production of learning and study material and in the discovery of new concepts and ideas.

3.1.2 Computer Go files

With the introduction of the internet also online Go communities started to develop. Several different incompatible protocols and file formats are in use today on the Go playing servers [17]. Their purpose is to make it easy to exchange digital game records by the internet. The first recording standards were developed for the purpose of publishing magazines or other printed documents, which made

those versatile for creating articles. These standards allowed the user to generate a diagram of the actual game, but also to enter commentaries and variations or markers insight the diagram so that a situation or a thought process could be explained. The most popular and used format at the moment is the SGF / "Smart Go File" [15] standard which was proposed by Anders Kierulf in 1990. Because of the simple tree based representation SGF files are extremely compact and uncommented game files rarely exceed a size of 5KB. But as before the SGF standard was originally designed for the exchange of Go records between humans and not for spreading computer oriented expert knowledge.

3.2 XGF an extension of SGF

When attempting to create standard to exchange expert knowledge compactness of the representation should still be considered significant when possible, however because of the ability to maintain larger amounts of data on computers the focus of developing a new standard should lie in:

1. Flexibility of adding different kind of low level and high-level features of different programs
2. Ability to incorporate (computer usable) descriptive expert terminology for game positions
3. Usability and easy extraction of features for none expert Go players
4. Ability to transfer data to SQL-databases and number manipulation engines like Matlab
5. Upgradeability of the standard by a new standard

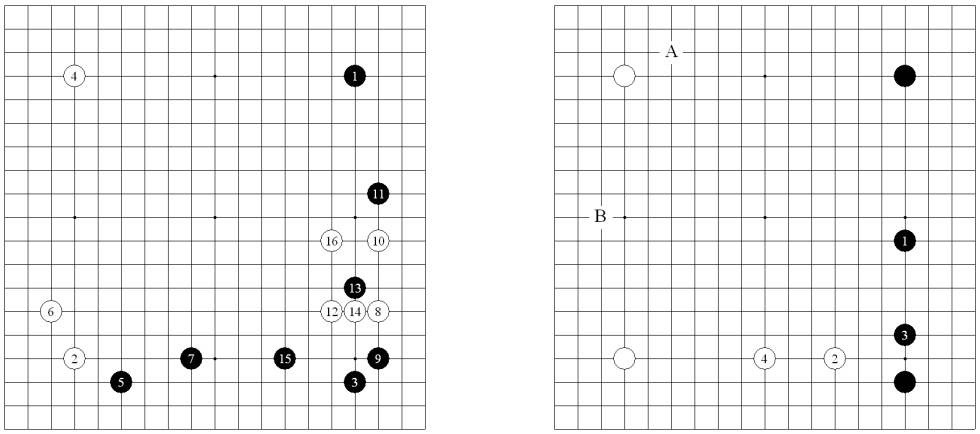
Of course it is also important to create this file standard under consideration of compactness, however compression tools like pkzip should deliver acceptable files sizes. The SGF standard provides a basis for most of the above mentioned points however SGF lags the ability to enter expert knowledge relevant for machine learning as well as portability to SQL databases and easy number presentations to Matlab. In 2002 Arno Hollosi [16] proposed the use of the XGF standard which was meant as an XML replacement for the existing SGF standard allowing therefore compatibility with XML databases. Because the SGF standard was already widely used and developers did not see the need for a new standard,

XGF has never been introduced as an official standard. Although XGF does not propose any methods to exchange expert knowledge other than the insertion of game commentaries, the principle of using XML should strongly be considered for developing of a new standard to exchange expert data. The ability to exchange expert data in combination with their relational models could create benefit for current and future researchers. A researcher who developed a certain new feature would then be able to improve an existing Go record by simply adding them with a description of their use and purpose. Recording features generated by the computer in such a standard would probably cause for unnecessary increase of the file size. The generated features for the experiments conducted in this thesis for instance are over 3GB in size and would be better shared by a common feature extraction tool.

3.3 How expert data has been recorded

As the generation of a new file standard would be beyond the scope of this thesis a middle way has been used to record and share the expert data using the current file standard SGF. A game of Go can be recorded by writing down the sequence of moves in the played order. SGF describes a tree based representation of a Go record where each move is inserted as a node within the tree. The nodes are separated by semicolons and a move is recorded with B[x] or W[x], standing for a white or black move respectively at point x. SGF also allows to enter annotations and variations. Annotations are recorded by adding additional properties to a node, for instance if we want to label a point x with the letter 'A' then we insert LB[x:A] at the respective node within the game tree. Variations are used to explain a situation by showing an alternative way of playing, see figure 3.2 for an example. Inside SGF variations are branches within the game tree and are entered by placing brackets at a node where the variation is supposed to appear. This also makes it possible to enter variations of variations. Although the standard allows to enter several different kind of annotations the standard is rather inflexible permitting the adding of relational information so that new annotations have been defined which can be used inside the current standard. The annotations can be positioned in the game tree like other annotations.

The portability of the information is not trivial and would require a programmer to make adjustments if he/she would decide to read the expert information with his/her current SGF file reader software. As it will become clear in chapter 5, the recorded expert information describes the relations between stones, chains or



(a) A game record played

(b) Variation: instead of 5 in the game record, Black can also choose to play 1 inside the variation resulting in the "Chinese opening" after 4 Black can play at A or B

Figure 3.2: Example of a variation with annotation

empty areas. Using first order logic this kind of relations can be easily described. For instance if we want to say that stone A and stone B are strongly connected we could generalize this with saying A and B have property p or also p(A,B). The arity of a relationship p should be flexible hence it is proposed to record such relationships as a new property inside the current SGF standard the following way:

$$p(\{a_1, ..., a_n\}, .., \{z_1, ..., z_n\}) = X_{identifier}[a_1, ..., a_n|...|z_1, ..., z_n|p]$$

It is suggested to group similar relations with an identifier and label the individual differences with p. For example, the connectivity between two chains has been recorded as follows:

$$XC[x|y|3]$$

Indicating that there is a connectivity relationship between chain x and y of category 3. The recorded relationship should be placed at the relevant node inside the game tree the same way other annotations are recorded. The merit of using a notation involving first order logic is to make the annotations also usable for Go projects which are using logic programming for predicting a next move. For this

thesis however these annotations have simply been used to extract expert labels crucial to the learning process.

The idea of the proposed extension to the SGF standard is to make it easier for researches to add expert knowledge using simple first order logic principles. This way more descriptive datasets can be generated and concept learning in Go can be stimulated for new researchers and programmers. For this thesis annotated Go records have been generated and stored in a SQL database for accessibility reasons. The Go records can be transferred to single SGF files and read using an adjusted SGF file reader. The produced expert data can therefore be considered self-contained.

Chapter 4

XiGo - tutor and learning program

For the purpose of this research project a Go editor and learning program has been developed: XiGo. The X is the Greek letter χ, hence is the pronunciation of the program *ch'i go*, *ch'i* standing for the concept of life force or spiritual energy in Chinese culture. Several free Go programs and editors even with available source are already freely available like GnuGo (written in C++) [12], GoGui (written in Java) [9], and the Tesuji Go Library (written in Java) [2], but for the requirements of this project writing a new editor instead of expanding old editors appeared to be the better choice. In the beginning, the focus of the research was creating a large database of Go problems with incorporated human expert knowledge, which is then used for experiments using modern learning algorithms. Both of these tasks needed to be fully implemented in existing software. Hence, writing a new GUI in addition with only the necessary functions seemed to be more attractive than trying to add new functionality to existing programs as exhaustive deepening into those programs would have been a prerequisite. Also by introducing first order logic inside Go records, having a proper editor in place would make the input process easier and has been taken into consideration when developing the software.

The result of the implementation is a Go program with three modules: XiGo Tutor, XiGo Xtractor and XiGo domain, see figure 4.1. XiGo Tutor is an editor which allows a Go tutor to enter the Go problems with expert annotations into a SQL database. The SQL database chosen is MySQL and is only used for accessibility purposes as each individual problem needs to be stored permanently in a single string. XiGo Xtractor is a program which is able to extract necessary

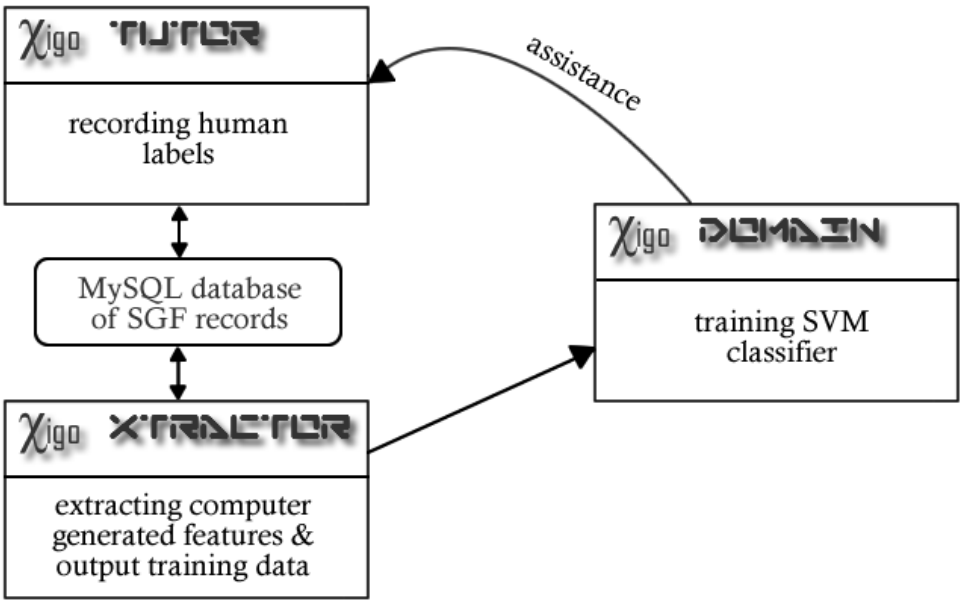


Figure 4.1: The XiGo software concept

computer generated features and expert labels derived from the annotations. The features in combination with the expert annotations or labels are then stored in a training data file. These are subsequently used to train a Support Vector Machine (SVM) a learning method used for classification explained in the next chapter. XiGo domain is the module where the actual training of the SVM is processed. Although not yet implemented, it is possible that the learned classifier can assist the tutoring module as discussed in the final chapter.

A short overview of the functionality of all three modules is given in the next sections.

4.1 XiGo Tutor

The XiGo Tutor allows for the input and display of expert data by a human expert, see figure 4.2 for snapshot. The module interacts directly with the SQL database of problems, which allows the user to browse through the whole dataset

conveniently. The extra value of the tutor compared with other editors lays in the ability to enter expert data. Next to the conventional annotations which can be inserted inside a game record such as simple labels like triangles and letters, is it also possible to enter the annotations specifically designed for this thesis. The annotation options can be selected at the left side of the XiGo Tutor panel. The user is allowed to enter a move sequence by selecting the *play* button at the right. Also, it is possible to record additional stones into a position with the button below the *play* button. The magnifier button allows to enter a *focus* on the Go board, which specifies what parts of the Go board are relevant by highlighting the area. The subsequent buttons *abc*, triangle, circle, square, and X are all SGF standard annotations which were hardly used. The last three buttons allow to enter three of the expert annotations designed for the thesis; namely: connectivity, vital points and eye space. All the three annotations are explained in the next section in detail. Other expert annotations recorded are the move type and outcome type of a problem, which can be selected with a drop down menu on the top of the panel. Other buttons allow the user to enter variations or browse through the database. The module has been set up such that additional expert annotations types can easily be added. It is possible that XiGo Tutor can assist the expert annotating by making suggestions derived from a generated classifier.

4.2 XiGo Xtractor

XiGo Xtractor is a module which interacts with the SQL database to retrieve each individual annotated Go record to generate features relevant to a learning task. In addition labels from the expert annotations are extracted which are then combined into training examples stored into a training file and one testing file. The testing file generated contained 10% of the whole extracted data, while the training file consisted of 90%. Because of constraints imposed by the SVM learning method scaling the data was also part of the extraction process. The whole procedure is automated and does not require the assistance of an expert. The generation of features can be time consuming, therefore the output of the extraction process is saved on the hard drive. A training file can have a size between 20MB to 1.4GB, hence the generation of a training file and the actual training of a classifier were independent processes, as hard drive space was not an issue. The Xtractor allows the user to export the example files in two kind of formats. One format is used by the XiGo domain module and the other format allows the direct use of the learning application libsvm [7], which is presented in the next chapter.

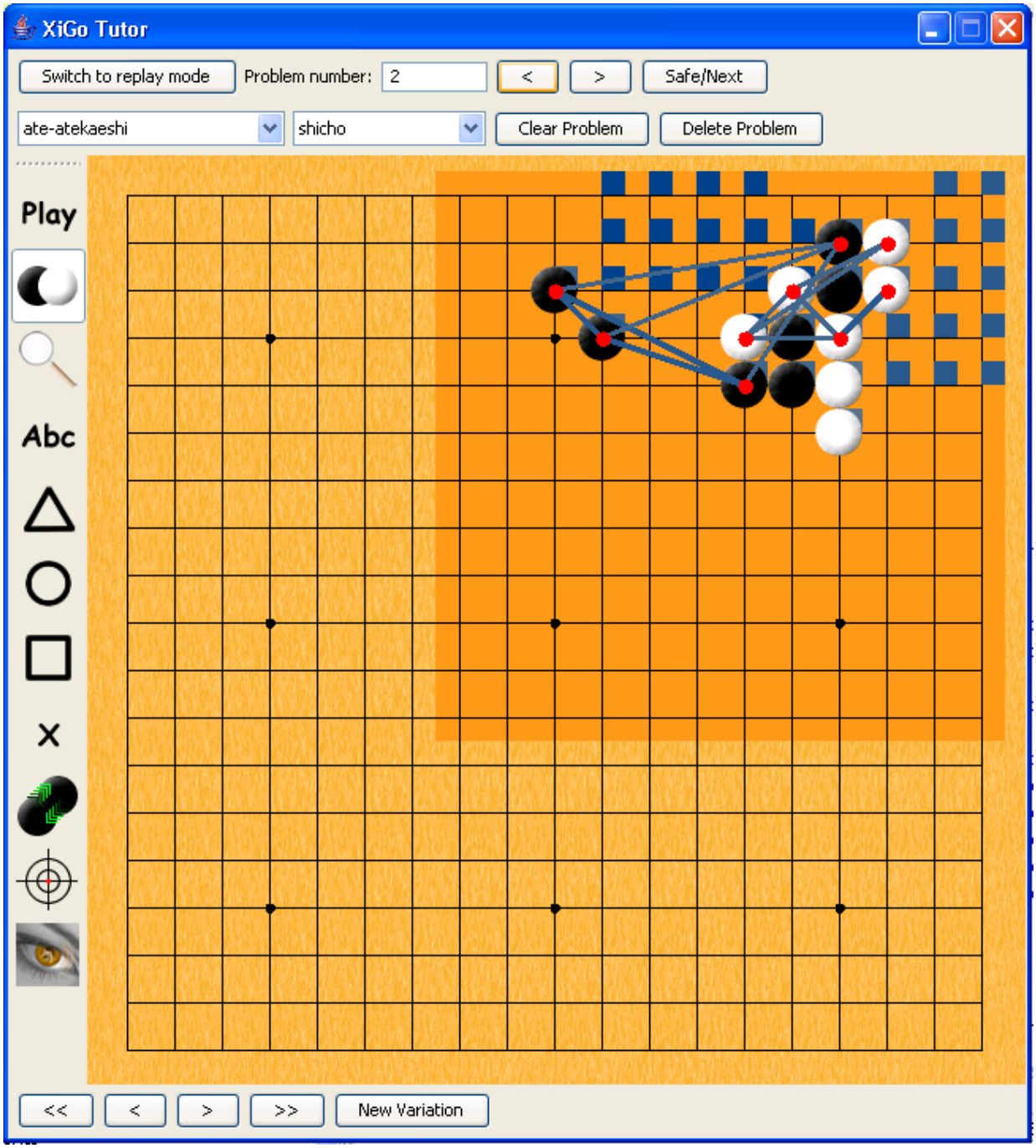


Figure 4.2: A screen shot of XiGo Tutor

4.3 XiGo domain

XiGo domain is the module of the program in which the interaction between training data and the learning application is done. The module's purpose is to generate a classifier for a selected expert concept recorded with annotations. In the setup chosen it is presenting the java implementation of libsvm with a training file, which then returns a classification model for the expert concept. The model is then tested by predicting the expert labels from the test set. The module will then output several key statistics as well as a list of test examples which have not been classified correctly. The C implementation of the libsvm software proved to be more powerful than its Java sibling, hence XiGo domain has not been used for the experiments presented in this paper. Instead the files generated specifically for the C implementation of libsvm have been used. As it is considered to generate an interactive process during the tutoring procedure between the expert and the program, the module can be used to assist the expert annotating. The concepts of this procedure are explained in the last chapter where other possible future work is discussed.

Chapter 5

Feature extraction and classification

Learning expert concepts in Go involves three main questions:

1. How can expert concepts be recorded?
2. What kind of computer generated features can be used to learn the expert concept?
3. What learning method can be used?

All these aspects of the learning task are explained and examined in the sections below. This chapter is divided into three main sections. The first section describes what kind of and how expert information has been recorded. The second section explains what representation has been chosen and how features can be extracted from the representation. The last section describes the learning method used in the experiments.

5.1 Labeling of positions by human experts

Before choosing what kinds of human expert information should be recorded, it should be investigated whether they fulfill certain constraints. Because an expert should input them and a computer should learn them, we are looking basically at two types of constraints. The expert constraints typically result in the upcoming of the following questions: Is it possible for an expert to input this information in a practical and efficient way? Does this information have practical relevance to

the decisions made by an expert to find the next move? The computer constraints address other questions like: is it possible to learn the information with lower level features, if not: can new low level features be created?

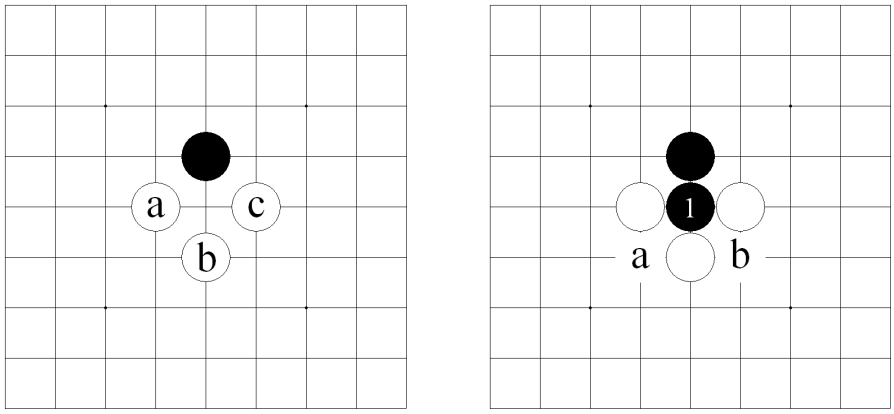
And there is of course the issue that different experts might have different opinions on the same situation, which raises the question whether this could be harmful to the learning process. Here, it should be taken into consideration that probably these differences are inevitable and that they likely allow a learning program to show some kind of individuality, as they can choose which opinion fits better with their created classifier. The opinion the system might choose to believe might fluctuate or even divert from both of the experts opinion, but this would be very similar to the human learning process and can only be desirable. A human expert who chooses a certain opinion might also divert from it in time as he might arrive to new conclusions in the matter concerned. But then again, if we do not mind the system to divert from the expert knowledge when can we say the system is able to learn the concept given? Here, it should be assumed that the expert opinion is likely to be true but not necessarily the exact one. So, if there is a difference between the system and the expert it should be within the range of one another.

5.1.1 Labeling the connectivity concept

Connectivity is an important and perhaps the most basic aspect in the game of Go. There is a classic Go proverb for beginners saying: "first cut, then think", suggesting that cutting opponent's stones usually will result in a favorable outcome. Often a game between two players is decided because a cut has split two important groups in two leaving behind an unmanageable situation for one player. Knowing where a cut can occur, also allows a player not to play careless and take preventive measures if necessary. Therefore, knowing what groups are connected and which ones are not, is crucial to understand a given situation. Often finding a cut for an expert player involves two stages. In the first stage, a cutting shape must be recognized. Then in the second stage the player would examine the situation by reading and confirm his findings. The recognition and analytic part go hand and hand, the more experienced the player is the more he realizes in the recognition stage than in the analytic one. Hence a beginner is more inclined to use his analytic ability to confirm his suspicions, often this leaves the player perplex in complicated situations as channelling ones analytic ability to the right spots is vital. This suggests that learning connectivity based on pattern

recognition should be a highly relevant area to apply Artificial Intelligence, as the involved parameters for finding an answer are far simpler than those required for finding the next best move for instance.

The concept of connection in Go is not as logical as it might seem to a layman. The principle of connectivity might easily suggest that there is transitivity, e.g. if $A \longleftrightarrow B$ and $B \longleftrightarrow C$ then also $A \longleftrightarrow C$. However this is not the case, a simple example shows us that we cannot reason with transitivity. In figure 5.1 the white stones a and b are inseparable because they can only be separated if Black would be allowed to play two moves. The stones b and c are also inseparable, however if Black plays at 1 in the right diagram then Black can either separate White by playing at a or b. It is for this kind of difficulties amongst others that computing connectivity using logical reasoning becomes a difficult task, which can be exploited by human players when playing a computer program. Tristan Cazenave and Bernard Helmstetter have addressed this specific issue using an analytic approach [6].



(a) White 'a and b' and 'b and c' are connected

(b) After Black plays 1, white cannot connect at both points a and b

Figure 5.1: An example of non transitive connections

The used labeling method:

Connectivity is a concept which describes a relationship between two same colored chains. Given a position a human expert can select a tuple of chains which he/she believes is relevant to the learning task and labels their connectivity. The expert has been given six different levels of connectivity to classify how strong or weak

the connection between two chains of the same color are. Each tuple of chains has been inspected from the player perspective of the chains' color. The opponent being the other color. The levels of connectivity given were:

1. The chains are *strongly connected*, meaning the chains cannot be separated even if the opponent would be able to play at least two moves after another in the area without interference, see figure 5.2 (a) and (b).

2. The chains are *connected*, meaning the chains can only be separated if the opponent would be allowed to play two moves in the same area without interference, see figure 5.2 (c) and (d).

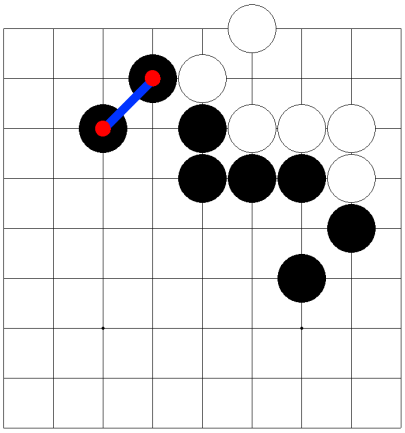
3. The chains are *conditionally connected*, meaning that the chains are only connected if the player would spend another move in that area. The chains could be separated if the opponent would be allowed to play in that area the next move, see figure 5.2 (e) and (f).

4. The chains are *separated*, meaning that the chains can only be connected if the player was able to play two moves without interference in the area, see figure 5.3 (a) and (b).

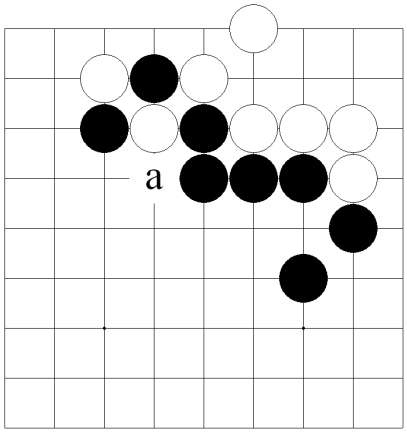
5. The chains are *strongly separated* if the player would be allowed to play three moves without interference in the respective area, see figure 5.3 (c) and (d).

6. The chains have *no connectivity*, means none of the other five connectivity level applies.

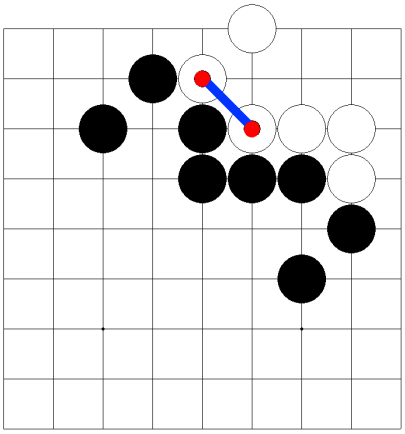
The expert would label the connectivity relationship of the two chains with the first five levels described above. If none of those 5 levels would apply, category 6 could be assumed. To label the connectivity between each same colored tuple of chains on the Go board is often not doable, as often the number of chains is too large. With 20 black chains for instance the expert would have to label $20*19/0.5 = 190$ different connectivity relations. Therefore, the expert was allowed to chose from his/her point of view relevant connections at his own discretion. This procedure led to about 11 connections in average per problem.



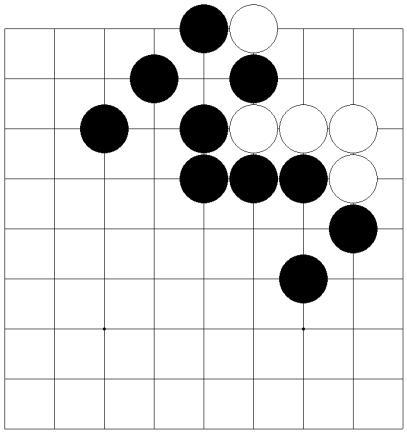
(a) The connectivity of the marked chains is *strongly connected*



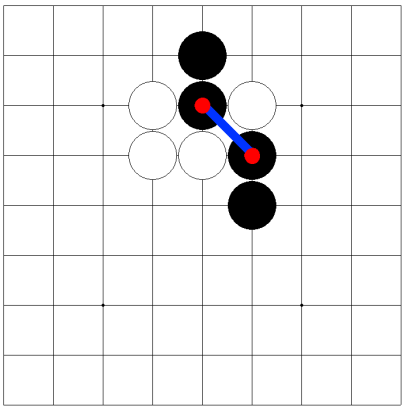
(b) Even if White was allowed to play two moves trying to separate the marked stones, Black can still capture one of the white stones by playing a.



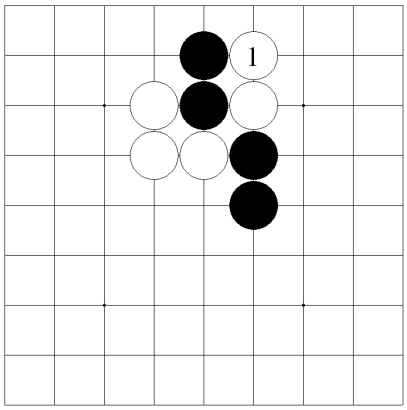
(c) The connectivity of the marked chains is *connected*



(d) The marked chains are clearly connected, however, not strongly connected as one of the stones can be captured with two moves.

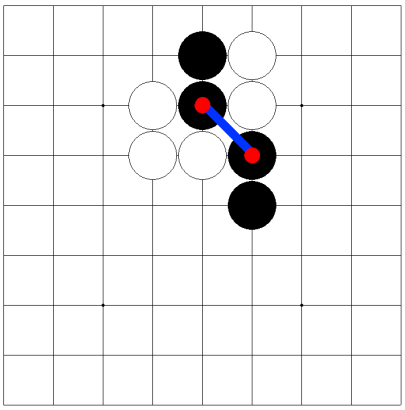


(e) The connectivity of the marked chains is *conditionally connected*

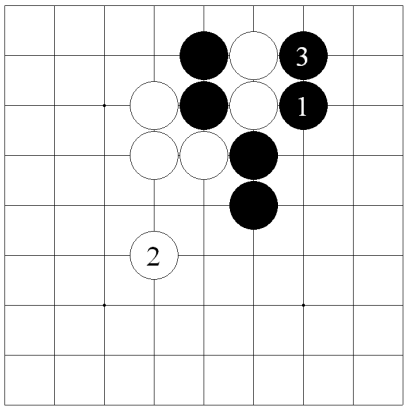


(f) If Black does not play, White can separate the black stones by playing at 1, capturing 2 stones.

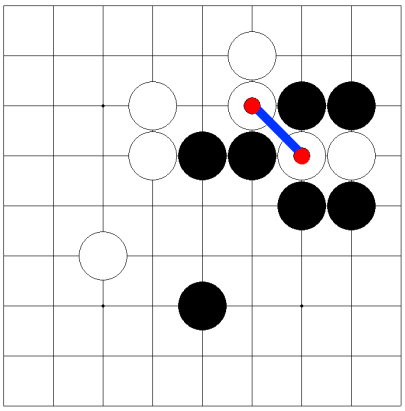
Figure 5.2: Examples of *strongly connected*, *connected* and *conditionally connected*



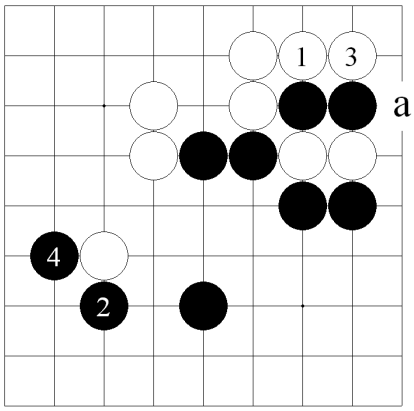
(a) The connectivity of the marked chains is *separated*



(b) The two chains can only be connected if White was allowed to play two moves undisturbed in the area



(c) The connectivity of the marked chains is *strongly separated*



(d) The two chains can only be connected if White was allowed to play three moves undisturbed in the area, capturing the black stones finally with 'a'

Figure 5.3: Examples of *separated* and *strongly separated*

Often considering the connection between two chains is rather dubious, because of their distance. Often an indirect (transitive) connection between two chains was omitted because of its irrelevance to the player. Note that also less trivial connections between chains have been entered, such as the connection of chains which are separated by another opponent chain which could be captured

playing the right move sequence, see figure 5.2 (e). Labeling this type of none trivial connections requires a minimum knowledge of tactical understanding of the game which makes learning a connectivity classifier more challenging.

The label for describing the connectivity was inserted with:

$XC[chainA|chainB|category]$

inside the SGF game records.

5.1.2 Labeling move and outcome type concepts

The entered positions are so called *tesuji* or best move problems. If Go could be compared to a boxing fight, then *tesuji* moves can be translated as knock out punches, meaning that their recognition results in totally different outcome than when not played. Every problem has been recorded along with the *tesuji* move, and a label for the type of move and outcome involved. In total 43 distinct move types and 45 distinct outcome types have been defined.

The labels for describing the move and outcome types were inserted with:

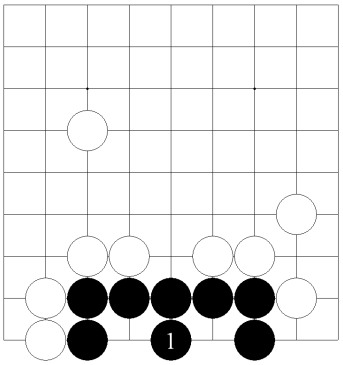
$GN[movetype|outcometype]$

at the beginning of a game tree inside the SGF game records.

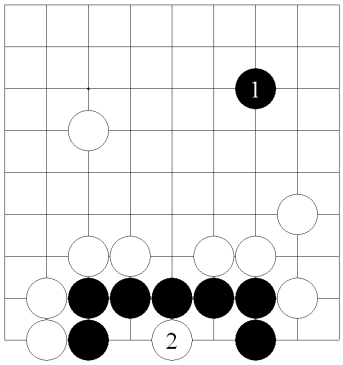
5.1.3 Extension: Labeling eyespace and vital points concepts

Eyespace plays a crucial role in Go. Again expert players rely on two skills, recognition and their analytic ability. In order for a group to live, it needs to have two eyes ("separated surrounded liberties"). To see whether a group or parts of it have two eyes is extremely complicated. To determine whether a group is alive or not, a player looks at the *eyespace* of the group, which can be described as an area with potential to create an eye. Sometimes, an *eyespace* can be big enough to allow the player to make two eyes by putting another move inside. In case the player would decide not to move there, then the opponent can remove one eye by putting a stone at the exact same spot. This kind of spot can be labeled a vital

point, see figure 5.4. Vital points usually are points on the Go board which are essential to both players which are not necessarily related to *eyespace* areas.



(a) Black plays and lives



(b) Black plays somewhere else and White kills the black group

Figure 5.4: An example of a vital point

From the description above it can be noticed that a vital point is not related to the color black or white, but applies to both. Eyespaces on the other hand always belong to a group on the board. In rare cases potential eyespace can be shared by both players or different groups from the same color, therefore, the expert needs to identify the concerned group of a classified eyespace as well. Seven categories for classifying these have been developed:

1. The eyespace contains a *strong living shape* even if the opponent would be playing twice.
2. The eyespace contains a *living shape* even if the opponent would be playing first.
3. The eyespace contains *two eyes* if the player would spend another move.
4. The eyespace contains a *solid eye* if the opponent would spend another two moves
5. The eyespace contains *strong one eye* even if the opponent would be playing first.
6. The eyespace contains *one eye* if the player would spend another move.

7. The eyespace contains an *unlikely eye* if the player would be allowed to play two moves in the same area without interference.

The labels for describing the eyespace have been inserted with:

$$XE[a_1, ..., a_n|chain_1, ..., chain_n|category]$$

inside the SGF game records, $a_1, ..., a_n$ denoting the empty intersections of the eyespace and $chain_1, ..., chain_n$ denoting the chains which relate to the eyespace.

For the vital points the following two categories have been developed and entered:

1. The vital point is a *life and death* deciding point.
2. The vital point is a *shape improving or teasing* point.

The labels for describing vital points have been inserted with:

$$XV[emptyintersections|category]$$

inside the SGF game records.

The eyespace and vital point labels were entered on an experimental level with the XiGo Tutor but were not extracted by the XiGo Xtractor, as no learning methods for theses concepts have been developed.

5.2 Representation and feature extraction

Learning a good classifier for a concept requires an adequate representation of the underlying problem of the concept. If the representation of a problem is well defined the learning task is expected to deliver results with higher accuracy. Here lies one of the biggest challenges in applying Artificial Intelligence to Go: What are relevant features for the learning tasks? Can we describe a situation in such a concise way that the learning of any concept can be done with that description? The answer is probably no. Instead of looking for a universal representation of the Go board, it ought to be wise to consider each learning task individually and select intuitively appropriate features relevant to the problem at hand. In the next subsections, several plausible feature representations are explored.

5.2.1 Chain features

Some of the most basic features of chains can simply be derived algorithmically. This features are:

- Number of stones.
- Number of liberties.
- Number of adjacent enemy chains.
- Number of adjacent enemy stones (sum of stones of adjacent enemy chains).
- Number of friendly chains which can be connected with one move.
- Maximum number of liberties which can be obtained by adding another stone to the chain (can be from 2 to -1).

Although these features of chains are relevant to the game, they describe a situation of a chain too abstractly. Some of these features are derived from the representations of the following subsection, hence the direct inclusion of these features was omitted for all learning experiments.

5.2.2 Common Fate Graph

A richer graph representation was introduced recently by Thore Graepel et al.[13]. The Common Fate Graph's (CFG) with Relative Subgraph Features (RSF) have already been successfully used in an attempt to learn graphs with Support Vector Machines in the game of Go. Liva Ralaivola et al. also used this representation in addition with other pattern enriched features [22]. The principle of the CFG lies in the observation that chains in a Go position share a common fate and stones of a chain can be summarized to a single node within a graph. To explain how this is done, a formal definition of the graph is given.

Firstly we note that a board position can be described by Full Graph Representation (FGR) which is an undirected graph $G_{FGR} = (P, E)$. $P = \{p_1, ..., p_{N_p}\}$ is a set nodes p_i representing each point on the Go board. Each node p has a label $l : P \longrightarrow \{black, white, empty\}$ which describes what is positioned at the node. The set $E = \{e_1, ..., e_{N_E}\}$ is the set of edges with $e_i \in \{\{p, p'\} : \quad p, p' \in P\}$ which describes the edge relations imposed by the grid of the Go board.

The CFG can now be derived from the FGR by transforming it. The transformation involves merging nodes which have the same color and removing resulting duplicate edges. A formal definition of this transformation is given below.

Given two nodes $p, p' \in P$ that are neighbors or $\{p, p'\} \in E$ and their labels non-empty and identical $l(p) = l(p') \neq empty$ perform the following transformation:

1. $P \longrightarrow P \setminus \{p'\}$ (merging nodes)
2. $E \longrightarrow E \setminus \{\{p', p''\} \in E\}) \cup \{\{p, p''\} : \{p', p''\} \in E\}$
   (this removes duplicate edges)

Applying the above transformation on the FGR until no same colored neighboring nodes exist will result in the CFG. An example transformation can be seen in figure 5.5 where a RSF in (a) is transformed to a CFG (b). The CFG is simpler than the FGR and has the advantage that nodes can be representative for chains, which makes the graph more suitable for learning tasks. The reason for that is that Go players simplify neighboring stones into a single entity too.

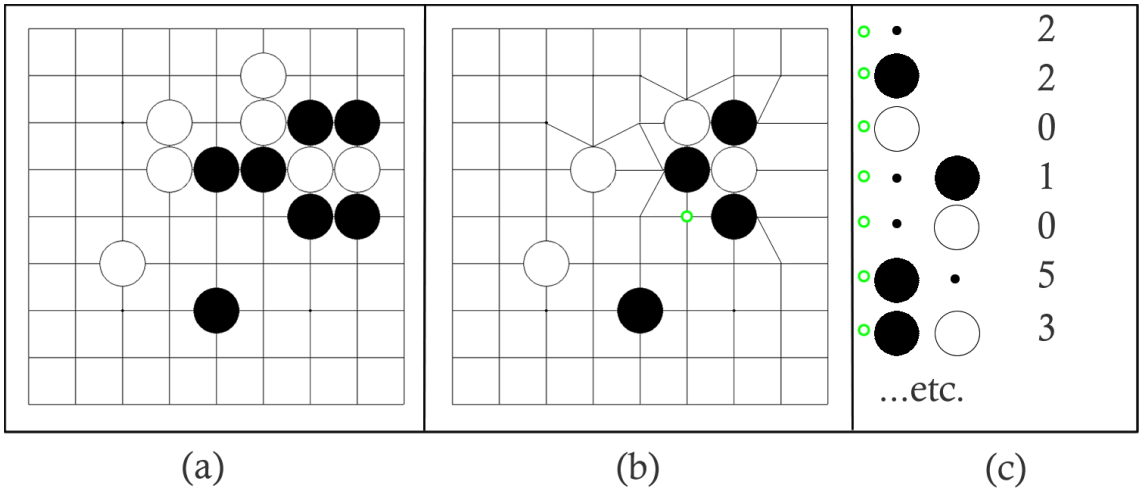


Figure 5.5: An illustration of the feature extraction process. In (a) the FGR representation is given, (b) is the derived CFG and (c) shows the RSF of the green marked node inside the CFG.

5.2.3 Relative Subgraph Features

The Relative Subgraph Features are used for learning the concepts of move type, and best local move.

The learning methods described in the next section operate on object representations called feature vectors $\overrightarrow{x} \in \mathbb{R}^d$. The question at hand therefore is: how do we extract feature vectors from CFG for our learning method? Or more practically: how can information be retrieved from the graph which can be adequately represented by a feature vector? Graepel et al. [13] propose the use of Relative Subgraph Features (RSF) which describe the pattern around a node in the graph. The RSF of a node are constructed with a mapping $\phi : CFG \times P \longrightarrow \mathbb{R}^d$ in the following way:

Let d be the number of possible connected subgraphs and $\tilde{G}_i = (\tilde{P}_i, \tilde{E}_i) \in CFG$ with $i = 1, ..., d$ such that $p \in \tilde{P}_i$. Further only select $\tilde{G}_i$ such that the subgraph has no branches and loops and that one of the ending points inside the subgraph equals p. The relative subgraph feature $x_i = \phi_i(p)$ is then the number of subgraphs $\tilde{G}_i$ which can be found in CFG.

While the formulation of the RSF might not give direct insight into what the RSF represents, it is helpful to look at a single feature for instance and how its value is obtained. Take the feature *empty* of the green marked node in figure 5.5 (b) for instance. This can be done by counting the number of adjacent empty nodes to the marked node which is 2. Another feature is *empty-black*. Starting again from the marked node the number of paths resulting in a *empty-black* are counted. In this particular case there is only one such traversal by going to the left and then diagonally to the upper right. The whole RSF describe the surroundings of a black or white chain or an empty space. In the conducted experiments only the RSF of empty nodes have been used. The length of a subgraph should be limited with a constant $s \geq |\tilde{P}_i|$ so that the size d of the feature vector remains in a reasonable range.

Intuitively the RSF is a description of the pattern around a node. This is similar to the image a Go player processes when seeing a situation. This led to the hypothesis that the RSF would contain sufficient information for learning the move type labeling. To put the hypothesis to the test an experiment involving the use of RSF to classify the move type has been made. Another hypothesis previously tried and confirmed by Graepel et al. is that the RSF contains enough information

to predict the quality of a move played at the respective node. The experiment for classifying the quality of a move by use of the RSF has been reproduced. Both experiments have been made using RSF vectors with a maximum length of $s = 6$ resulting in a maximum of 406 features.

As the strength of the above representations has been already pointed out, it seems useful to look at the drawbacks of the representation as well. A weakness of RSF is that the evaluation of a *empty-black-empty* feature might be misleading as it is unclear how powerful or weak the black node in the subgraph is. On the other hand this kind of information could be derived by looking at other more far reaching features. But it is conceivable that two completely different situations might result in a identical RSF vector.

5.2.4 Relative Subgraph Path Features

One of the main contributions of this thesis is the development of a new feature representation which we call Relative Subgraph Path Features (RSPF). The RSPF have been developed for learning the concept of connectivity.

While the Relative Subgraph Features describe the surrounding of an empty spot or a chain on the Go board, it says little about the connectivity between two chains. The richness of the CFG allows for other ways to exploit feature extraction. The simplest way to look at the connection between two nodes in a graph is by looking at the paths in the graph leading from chain A to chain B. As it is not sufficient to describe the complexity of a situation by looking at the shortest path from A to B, all possible paths from A to B are considered. The types of paths can then be counted the same way RSF are extracted. The hypothesis is that the developed Relative Subgraph Path Feature vectors would describe the connectivity between the chains sufficiently.

The procedure to extract the RSP feature is the following:

- Step 1: Consider two unconnected chains A and B of the same color.
- Step 2: Start at node A in the graph and then traverse all neighboring nodes.
- Step 3: Then from each node traverse all non visited neighboring nodes until chain B has been reached otherwise repeat step 3.
- Step 4: When chain B is reached a path or subgraph is obtained. The starting node of chain A and ending node of chain B inside the obtained subgraph are irrelevant as they are always of the same color, hence only the

nodes in between are recorded subgraphs. For each distinct subgraph path a feature will then be recorded the same way it is done with RSF.

As the feature *empty-black-white-empty* is technically different from *empty-white-black-empty* obtaining the RSP features by traversing from B to A delivers a different feature vector than traversing from A to B, however, their evaluation should be identical. The RSPF from A to B, $\overrightarrow{RSP}_{(A,B)}$, has also a one on one mapping to the RSPF from B to A, $\overrightarrow{RSP}_{(B,A)}$, indicating that their evaluation would not differ. Intuitively, the connectivity between two chains would be best described by the sum $\overrightarrow{RSPB}_{(A,B)} = \overrightarrow{RSP}_{(A,B)} + \overrightarrow{RSP}_{(B,A)}$. The mapping to RSPB is injective, suggesting that information could be lost by using it. But it could also imply that new information is generated, making RSPB richer in description. To investigate which path representation is most useful for the learning task the following 3 cases have been investigated and compared:

1. RSPF type A: only features from the paths from chain A to chain B are used, see figure 5.6 (b).
2. RSPF type B: (RSPB) the features from the paths from chain A to chain B and vice versa have been used and added up in a single feature vector, see figure 5.6 (c).
3. RSPF type C: the features from the paths from chain A to B and vice versa are used in two different feature vectors generating twice the amount of training samples. This would cover the feature space more sparsely than type A.

In fact one could consider another type by appending the feature vector from chain B to chain A to RSPF type A doubling the feature space. However, for this type there would be a direct one-on-one mapping with RSPF type A making the considered type identical to RSPF type A. Which of the types is most suitable for learning connections is addressed in the experiments and results chapter. The maximum path length used in the experiments is $s = 8$, which is a sufficient length for describing the connectivity relationship for the Go board.

5.3 Support Vector Machine classifier

Given human labels and computer generated features, it is possible to train a classifier which can predict the label of a new position. The used classifier in

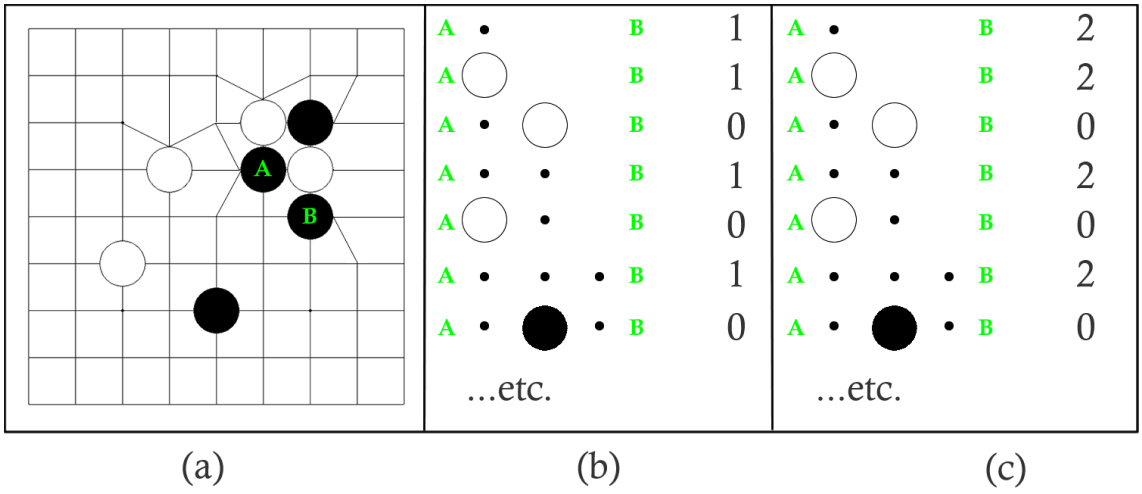


Figure 5.6: An illustration of the RSPF extraction process. In (a) the CFG representation is given, (b) is the derived RSPF type A of the nodes A and B and (c) shows the RSPF type B.

all experiments is the Support Vector Machine [5], which is a useful classification method. The method has been chosen as Graepel et al. showed that SVMs generate better predictors learning RSF vectors compared to kernel perceptrons [13]. The essence of the SVMs lies in the formulation of a mathematical problem. Given a set of labeled training instances $(x_i, y_i), i = 1, ..., l$, x_i being the description of the instance in a n-dimensional space $x_i \in \mathbb{R}^n$ (here the computer generated features extracted for the specific task) and y_i being the labeling of the instance with $y \in \{1, -1\}^l$ (here the labeling given by the expert), SVMs attempt to find a solution to the following optimization problem:

$$\min_{w,b,\xi}[\frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i], \tag{5.1}$$

subject to

$$y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0 \tag{5.2}$$

The description of the instances x_i are mapped into a possibly higher or even infinite dimensional space by $\phi(x_i)$. Predictions of the classifier for a point x are

then made by the decision function:

$$\hat{y}(x) = sgn(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b) \tag{5.3}$$

Where $K(x_i, x)$ is a kernel function of vertices. The kernel used in the experiments is the radial basis function (RBF), as proposed by [7].

$$K(x, x') = \exp(-\gamma \|x - x'\|^2), \quad \gamma > 0 \tag{5.4}$$

The learning task of a SVM can best be described with a small illustration in a two dimensional feature space, see figure 5.7. The learning task lies in the finding of a separating plane of two different labeled entities. In the example a simple line has been found.

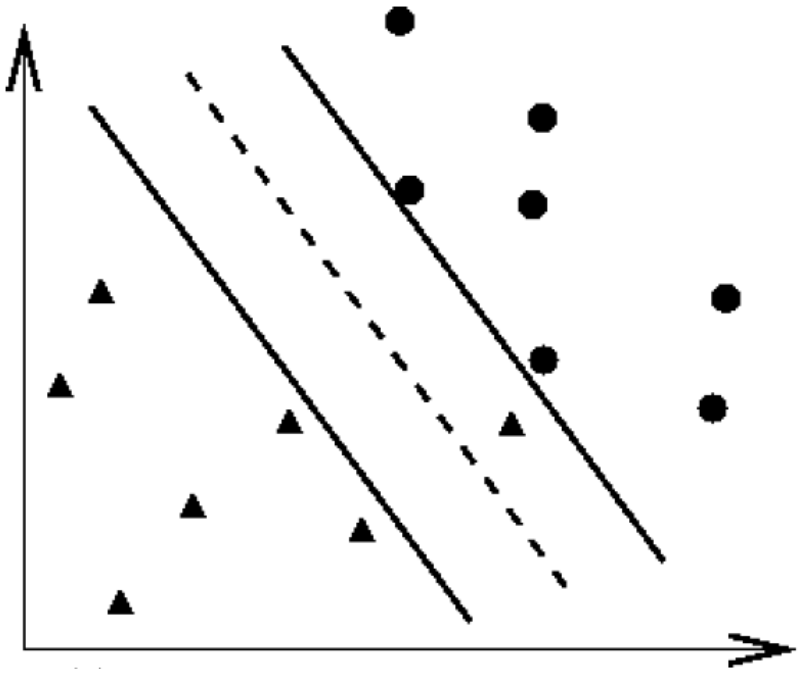


Figure 5.7: A generated classifier for predicting ■ and ▲ inside a 2 dimensional feature space

As SVM's have been widely standardized as a classification method, a variety of ready-to-use SVMs are available on the Internet [18]. The SVM used in the experiments is *libsvm* developed by Chih-Chung Chang and Chih-Jen Lin [7]. The SVM has been chosen because a Java interface was integrated in the software, which allowed the XiGo program to communicate with the SVM directly.

Chapter 6

Experiments and results

In this chapter an overview of the generated data and the description of the experiments with their results are presented. The first section describes what kind of Go records and expert labels have been recorded. The following sections then describe how this data in combination with computer generated features has been used to generate predictors. The predictors made during the experiments where connectivity, move type and best move predictors.

6.1 General setup of experiments

For the experiments a collection of expert-annotated Go records has been created. The collection consists of of 2,638 so called *tesuji* problems, or best move problems. The emphasis of these problems lies in finding the best local move in the given position. The problems are either given in a whole, half or quarter board context. There are 1,997 quarter board, 480 half board and 161 whole board problems. Each problem has been recorded with a solution and possibly with a variation on the correct or the wrong answer. In addition, all problems have been labeled with an outcome and move type if the best move is played. In total, there are 43 move type categories given ranging from simple diagonal move to complex move sequences. For the description of the final outcome 45 categories were recorded, such as "killing a group", "connecting", "sacrifice" or "torturing".

6.1.1 Recorded expert annotations

With the developed expert concepts 300 problems have been analyzed and labeled with annotations. The connectivity expert annotations proved to be the most

relevant to this dataset. In total 3,143 connections have been labeled averaging roughly 10 connections per problem. Some categories where better represented than others as they would occur more often, see figure 6.1. Especially the lack of category 4 and 5 made identifying those categories more difficult as can be seen later in the results sections.

Connectivity category	number of samples
1	422
2	1,472
3	957
4	228
5	64

Figure 6.1: Number of samples per connectivity category

The recording of eyeshape concepts proved to be irrelevant for the dataset as most problems do not involve life and death tasks. However, in total 140 eyeshapes have been identified which could be used for future learning tasks. Locating and labeling vital points proved to be the most irrelevant concept developed for this data set and only 15 vital points have been labeled. The concept of vital points could probably be improved by defining a vital area as a set of best black moves and best white moves related to the same problem spotted on the Go board. Often, one player would play in an area slightly different depending on the global conditions, as a position can actually have several vital points.

6.1.2 Testing methods and results presentation

In all experiments SVMs have been used to predict the expert labeling. Scaling the training data before applying SVMs is very important [24] and has been done by dividing each instance of a training vector by the maximum value it has in the entire training set. A SVM can only create a good model when the parameters γ and C have been chosen optimally [7]. For this reason, all experiments determined this parameter by doing a grid search over a part of the γ, C space. The accuracy of a possible classification model would be then predicted using 5-fold cross validation. In 5-fold cross validation the training set is split into 5 similar sized subsets. A classifier is then learnt by leaving out one set at a time, with which a temporary classifier would be validated. This method is considered the appropriate method for finding correct γ, C values, as finding these values by optimizing the results for

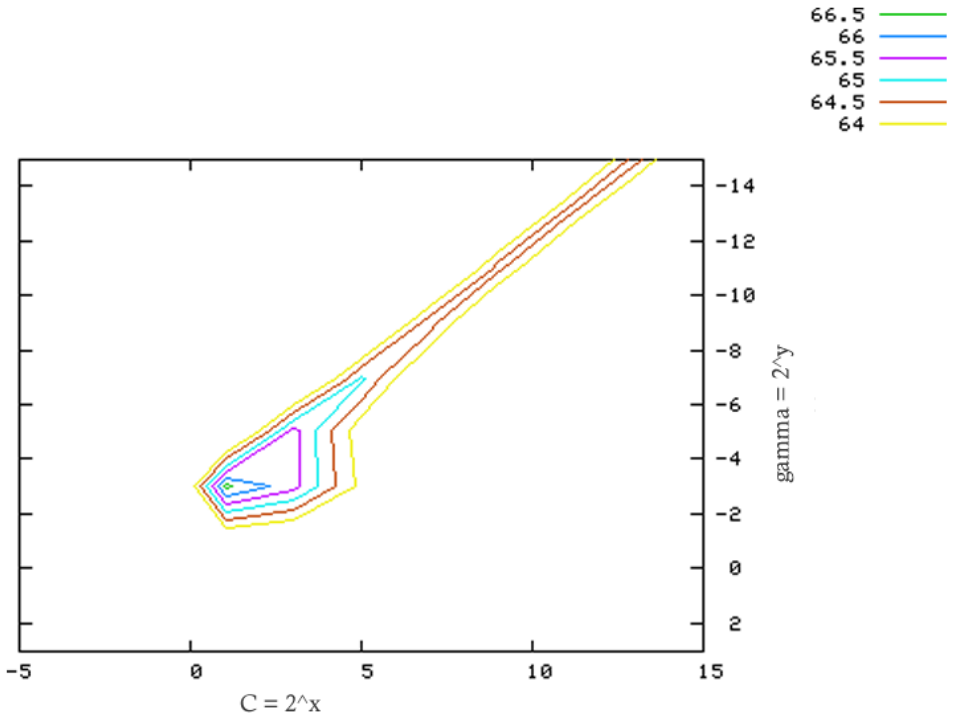
a separate testing set could result in overfitting of the classifier in respect to the testing set [7]. The accuracy inside the γ, C space is described with 2 dimensional graphs marking areas with their validation-rates. The higher the rates are the higher the accuracy of a possible model. The optimal found γ, C are then used to learn a classification model which is then tested on a separate test set which is about one ninth the size of the training set.

6.2 Predicting Connectivity

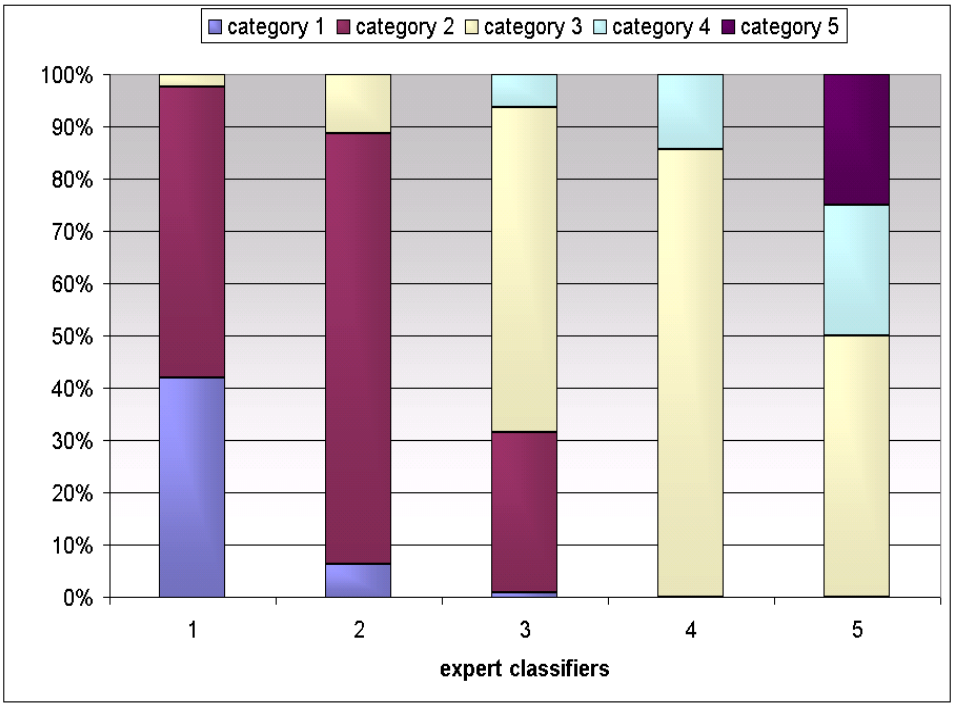
For each of the in chapter 5.2.4 defined relative subgraph feature types a separate test has been executed to determine which of the representation would be most suitable for the learning task. For type A and B identical training and testing sets have been used, such that each training example from the type A at any position within the data set correspondents with the training example of type B at the same position. The size of all RSPF types used was $s = 8$ as that would also be considered to be a maximum distance for a connectivity relationship between two chains on the actual Go board. Type C allows for twice as many training and testing samples so direct comparison of the test results could be misleading. As intuitively assumed before, Type C proves to be the most effective representation for obtaining accurate predictions. Type A proved to be the least accurate representation of the learning task as its optimal classifier showed a higher classification error rate than Type B and C. This leads to following conclusions. Type B features contain more relevant information than type A features, while type C features allow covering the feature space more sparsely. For future research it is therefore recommended to test type B and C for model selection. The results for each individual RSPF type are presented in the following subsections.

6.2.1 Relative Subgraph Path Features type A

RSPF type A showed a maximum rate of 66.5% correct classification in the grid search with cross validation at $\gamma = 0.125$ and $C = 2$. Meaning that the generated classifier was expected to predict the test set with a 66.5% success rate.



(a) Grid search results for RSPF type A



(b) Results on testset for RSPF type A

Figure 6.2: Test results for RSPF type A

The detailed findings of the grid search can be seen in figure 6.2 (a).

With the found parameters, a model has been learned with the SVM for predicting the expert labeling. The test results showed that the model was able to predict the correct category with 66.0% accuracy with a mean squared error of 0.378% and a correlation of $R^2 = 0.476$. Figure 6.2 (b) shows the evaluation for the individual categories. Each bar represents a class classified by the expert and its coloring reflects how well the class has been predicted by the classifier. It can be concluded that the predictor had the biggest difficulty with learning the concept of separation. These results must probably largely be contributed to the small amount of available training data from those categories. Also the distinction between strongly connected and connected appears to be difficult. From the Go perspective this indifference of the classifier can be accepted as the difference is rather subtle for most decisions made during the game.

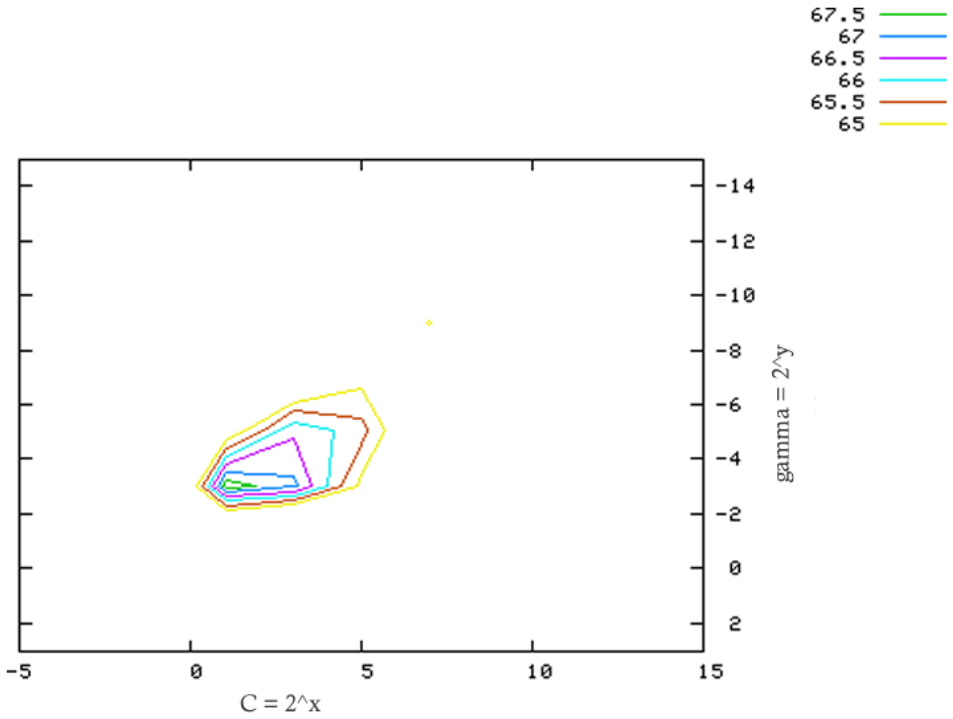
6.2.2 Relative Subgraph Path Features type B

As already pointed out RSPF type B proved to be superior to RSPF type A as can be seen from both the grid search using cross validation, as well from the comparison of the results from predicting the test set. The optimal parameters for the type B learning task happen to be the same as type A with $C = 2$ and $\gamma = 0.125$, see also figure 6.3 (a). The 5-fold cross validation with these parameters showed that an accuracy of 67.8% can be expected, which is slightly better than type A.

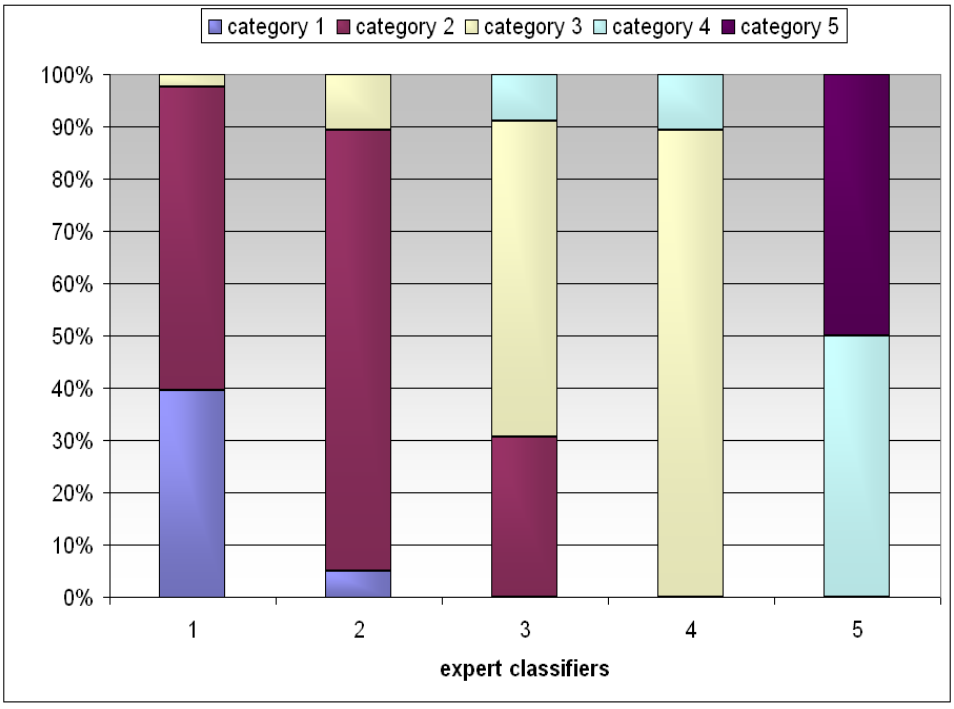
The test results delivered the same number of correct predictions as type A with 66.0% of the total testing set. The mean squared error $\mu = 0.349$ and $R^2 = 0.516$ were slightly better than type A as the predictions of wrongly classified test samples were in better proximity to their supposed category. The difference can be noted for the category 3 to 5. Yet those categories have not been learnt appropriately as the separate category 4 for example has been mostly predicted as conditionally connected by the model, see figure 6.3 (b).

6.2.3 Relative Subgraph Path type C

The best results were obtained with RSPF type C. Not only did the grid search using cross validation indicate that classifiers with higher accuracy up to 68.4% could exist, but also the results with the test set showed that the classifier was able to learn the concept of connectivity better than RSPF A and B.



(a) Grid search results for RSPF type B



(b) Results on testset for RSPF type B

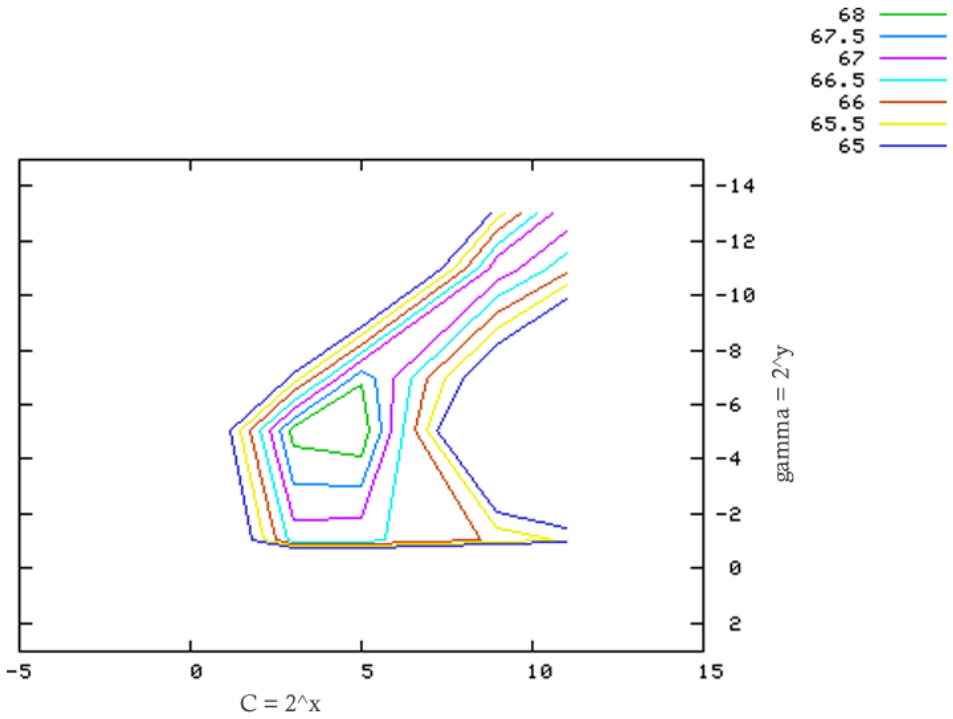
Figure 6.3: Test results for RSPF type B

The found learning parameters were slightly different from the ones found for type A and B with $C = 32$ and $\gamma = 0.03125$. The grid search results from figure 6.4 (a) also show that the area for parameters resulting in 65% accuracy is substantially larger than for type A and B making it easier to find good parameters. This could be vital if the training set might expand in the future as finding optimal γ, C values is a time consuming process. For the RSPF type C experiment for instance, a Pentium 4 with a 3Ghz CPU and 512MB RAM required roughly 7 hours to complete the grid search.

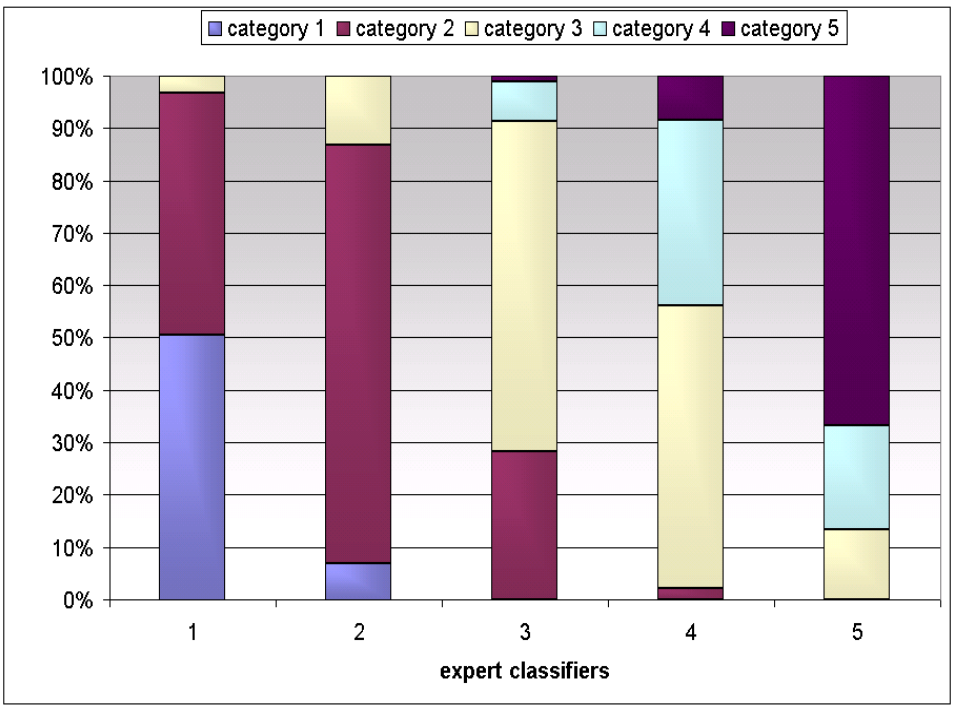
The test results shown in figure 6.4 (b) are the most accurate predictions generated by a model for connectivity. With a mean square error of 0.369 and a correlation of $R^2 = 0.5797$ a promising classifier has been generated. The class which proved to be the most difficult to learn was again the separate class. Interestingly connections labeled by an expert as conditionally connected have been predicted to be separated as well. This can be explained by the fact that some examples in the training set are situations in which stones appear to be separated although they can be connected if the player will capture the cutting stones.

6.2.4 Progress with number of samples

In the above experiments the SVM has learnt the connectivity concept with 2,828 (or 5,656 for type C) samples. The results for those experiments have shown a promising idea so far, although one should admit that the results also show that the classification model is not perfect yet. The question that naturally rises now is: what kind of progress can be expected if the training set size increases? To get an impression on the strength of the classifier in relationship to the number of available training samples, the experiment for the relative subgraph path type B has been repeated for different training sizes. With help of a γ, C gridsearch using 5-fold cross validation optimal learning parameters have been identified with which a SVM classification model has been computed. The model was then used to classify a testing set of 315 different samples not included in the training data. For each experiment the percentage of correct classifications, the mean squared error, and the correlation coefficient R^2 has been recorded and are shown in figure 6.5. The number of samples are represented on the x-axis using the logarithmic scale.



(a) Grid search results for RSPF type C



(b) Results on testset for RSPF type C

Figure 6.4: Test results for RSPF type C

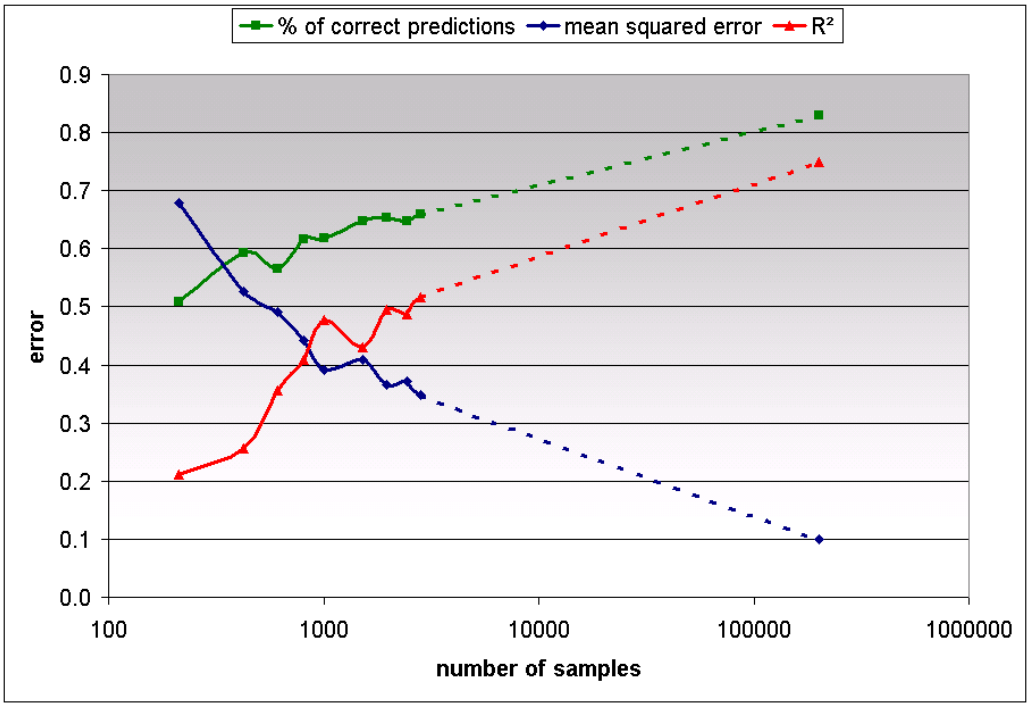


Figure 6.5: Correlation and error rates by number of learnt samples

The correct classification percentage is the number of correct predictions divided by the number of total predictions. The mean squared error tells about the difference between the wrong prediction and the expected answer, the lower the error the better. The last graph inside the figure is the correlation coefficient R^2, which also says how related the predictions with the intended outcome are. From the graph it can be seen that all values improve with the number of samples. Also, it can be derived that improvement of the classification model will require an increasing number of samples. The exact relationship between the number of samples and the quality of a learnt model is still inconclusive as sparsity over the feature space also plays a vital role for obtaining a good model. But estimating that about 200,000 samples are necessary for a prediction accuracy of about 85% by a learnt model is a tentative forecast. It should be noted that calculating such a model will demand a tremendous amount of computing time with an ordinary PC using the same learning methods as in this experiment. It is therefore recommended to consider alternative or improved classification methods to save computing time. The large number of training samples required for a high-level connectivity model appears to be rather large, but probably corresponds with what

an expert player would process before reaching expert level. One should consider that a single game easily involves around 200 connectivity evaluations and that an expert would play an estimated 1,000 games before reaching the expert level.

6.3 Predicting move type

The dataset has been conveniently recorded with what type of move the best move is. As one of the strength of the relative subgraph feature is its ability to resemble the pattern involved around an empty intersection, a correlation between the classified move type of a best move and its RSF vector is to be expected. Therefore, an experiment has been conducted to find whether the RSF vector of the best move can predict the move type category.

A support vector machine has been trained learning the movetype categories with help of RSF vectors of size $s = 6$. The training data for this experiment consisted of 2,374 samples, one for each tesuji problem recorded. The results of the grid search with 5-fold cross validation can be seen in figure A.3.

The results show an estimated 64.5 % accuracy within the grid examined. A model with the found parameters $\gamma = 0.5$, and $C = 8$ has been developed and tried on the separate testing set. The results confirm the percentage found by cross validation and suggest correlation between the RSF vector and the move type involved. Considering the small sample size for some of the individual categories, which can be rather similar as well, the results show that the RSF vector of an empty intersection can predict the move type involved. This suggests that the RSF vector contains a considerable amount of vital shape related information and has the ability to capture similar shape concepts as humans do. The result justifies therefore the use of RSF vectors for learning shape and pattern related concepts.

6.4 Predicting the best move

One of the experiments conducted by Graepel et al. [13] uses RSF vectors to learn the distinction between good and bad moves with computer generated Go problems on the 9x9 board developed Wolf [31]. The experiment showed that the classifier was able to discriminate between two offered moves one being the best move and the other being the bad move with a success rate of 85%. The problem set by Wolf is from a Go perspective rather unconventional and involves little pattern recognition skills for finding a solution, which makes the success

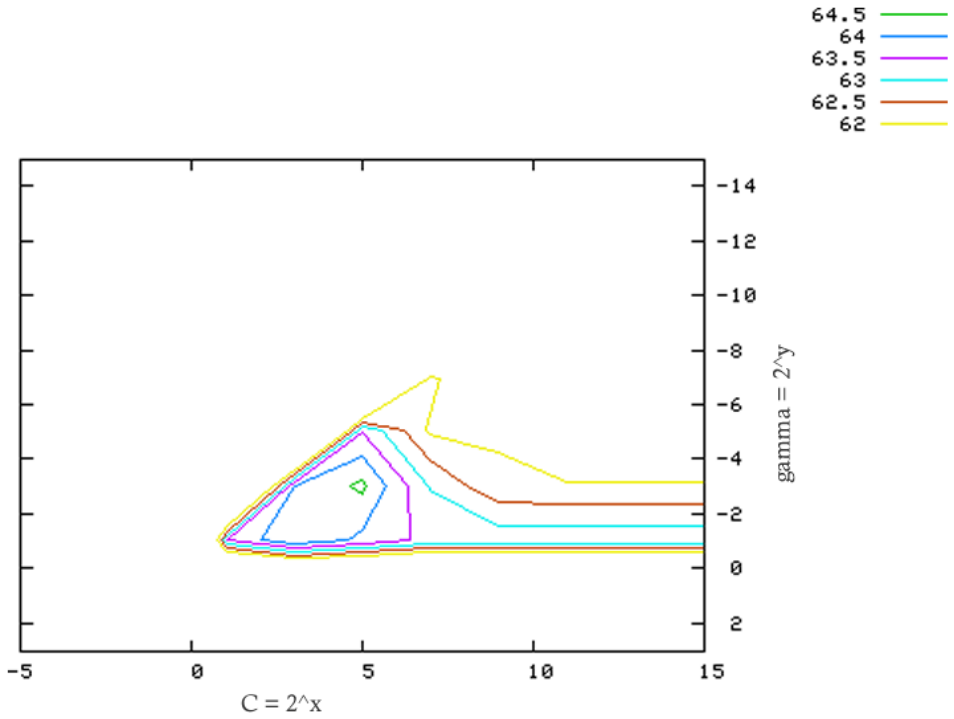


Figure 6.6: Grid search results for move type experiment

rate rather surprising. To see whether a similar success rate can be achieved with likely game appearing situations a similar experiment has been done on the *tesuji* dataset. The training set would consist of 2,374 RSF vectors extracted from the solution of each problem and an equal amount of RSF vectors from randomly selected bad moves. A set of 526 good and bad move samples would then be used for testing. Two distinct learning experiments have been made. One using the only the RSF vectors of size $s = 6$ for learning good and bad moves and another experiment using RSF vectors in combination with the move type and outcome information provided by the *tesuji* data set. The results are shown in the following subsections.

6.4.1 Relative Subgraph Features

The optimal found parameters for the SVM are $\gamma = 32$ and $C = 0.125$ with a success rate of 90.6% using 5-fold cross validation, see figure 6.7. On the testing

set the learnt model predicted the better move in 91.1% of the examples as well, which proves to be better than the similar experiment conducted by Graepel et al. using the computer generated problem set. The reason for this can be two-fold. The test results of this experiment are better because the training data is more suitable for the learning task or the distances between bad and good moves were bigger than in the previous experiments. Either way, this experiment shows that a discriminator for finding good and bad moves using RSF vector is feasible for finding answers to pattern related problems like *tesuji*.

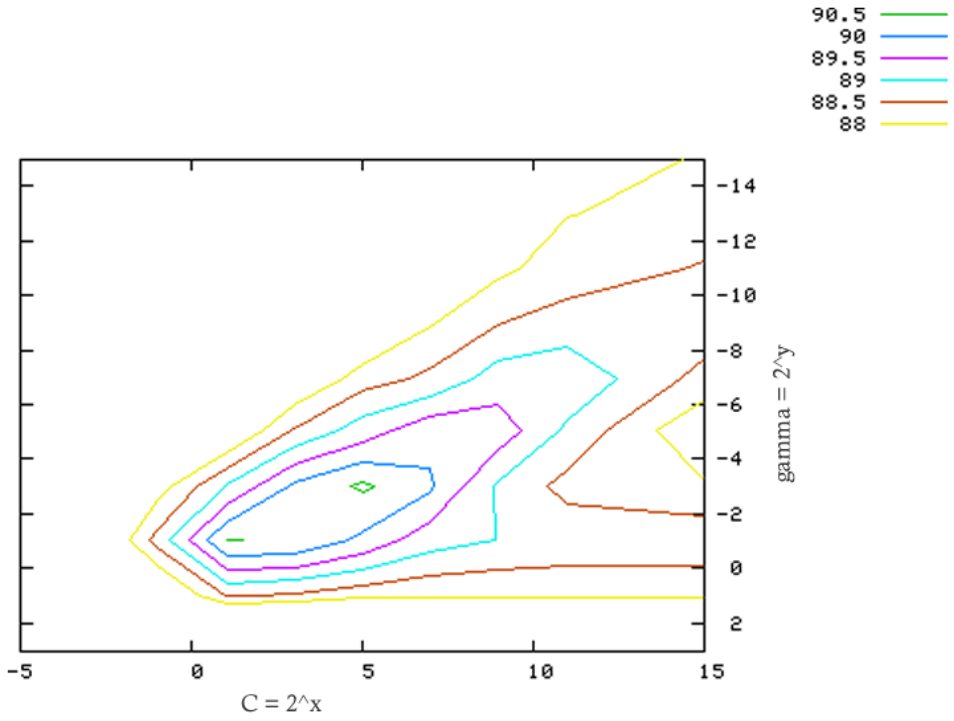


Figure 6.7: Grid search results for best move experiment

6.4.2 Relative Subgraph Features combined with move and outcome type labels

In addition to the above experiment a similar experiment has been made to see whether it could be helpful to hint what kind of best move the SVM should be

looking for by extending the RSF vector with information on the move type as well as the kind of outcome which can be expected when playing the correct move. The results show a similar but slightly better success rate compared to the previous experiment. With $\gamma = 8$ and $C = 0.5$ a success rate of 91% of the model has been found, while the test set has been predicted correctly 91.3% of the time. Again this results shows that pattern related information can be retrieved from RSF vectors.

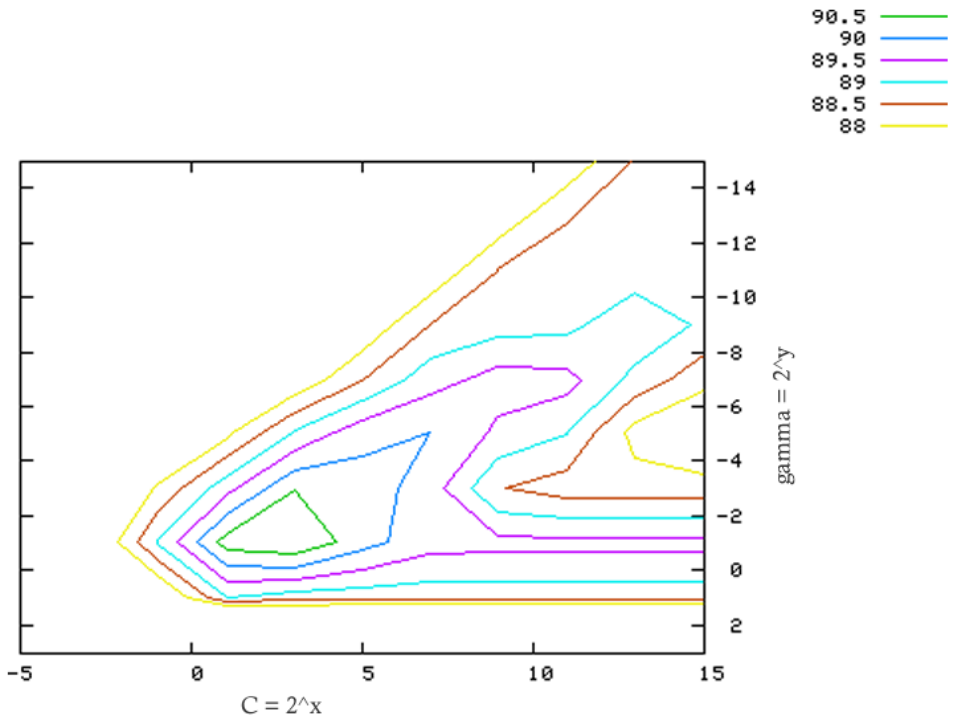


Figure 6.8: Grid search results for best move experiment using RSF, move and outcome type labels

Chapter 7

Discussion and future work

Learning whether two chains are connected on the Go board, might not seem a big breakthrough for the experienced Go programmer, as for them it might seem more natural to use conventional methods to solve such a problem. But on the other hand it shows a way for researchers how Go can be tackled by solving smaller problems with expert data. It should be mentioned that many of the learnt connectivity examples are not trivial even for the expert player, which suggests that advanced tactical skills have been learnt as well. Interestingly the learned classifier can generate a weighted connectivity graph of all the chains on the Go board which suggests that a similar graph orientated approach can be used to learn the more abstract concept of a group. By developing more and more of these classifiers from expert data, programs can learn the same high-level concepts as human players. Other areas where high-level classifiers could be developed are influence, territory and internal group strength. The art of letting these classifiers interact will then be the next step to develop a Go learning program. As has already been pointed out above, the next issue which will rise inevitably is the question of how the classifiers can be combined and whether these classifiers can learn from each other. In a sophisticated hierarchical approach low level classifiers could be adjusted from high-level reinforcement signals. Once a system would reach such a complex level of sophistication learning from professional games could lead to a dramatic improvement compared to the system used at this moment.

7.1 Influence, Strength, and other annotations

Next to the developed annotations one could think of many more concepts one could enter in a Go record to add game understanding information. Looking at

how a player reasons when analyzing a game with other players can be a good inspiration for the creation of such concepts. Of course the difficulty of creating an expert concept always lies in the representation of the concept and how it can be quantified in a sound and logical way. Other ideas for creating expert annotations could emerge by looking at the following concepts:

- *Strength* is a concept a player uses to asses how likely a group could be compromised by being attacked. Weak groups on the Go board usually suggest that a player should be alert not to create more weak groups. Labeling whether a group is weak or strong is rather arbitrary and intuitive. An expert could probably best describe the strength of a group with a scale from one to ten.

- *Influence* is another high-level concept used a lot by players to make their decision. It usually indicates areas where one color dominates and at the same time the player is not inclined to create territory in that area. Usually this also means the player would use his influence to attack or invade opponent groups.

- *Significance* is a concept players use to determine what areas or stones are more important than others and where action is required and where is not. One might feel this is closely related to the concept of territory, which is true, but many times this also involves cutting situations.

Current computer programs are likely using some of the above concepts in their code. But applying these concepts requires deep knowledge of a situation. The lack of this knowledge in the programs usually results in wrong interpretations of these concepts.

Learning of all of the above classifiers would require a richer representation of the game than currently available. Therefore, it would be premature to try to learn such concepts before actually being in possession of a representation which contains sufficient information for the learning task.

Another way to tackle the problem of learning patterns in Go could be by looking at the issue at a yet a smaller level of problems. The concept of connections in Go is actually very complex as high-level concepts go in the game. Finding the connectivity between two chains can be subject to many tactical influences. To simplify, grouping certain patterns or move types can be considered. For instance shapes like *monkey jumps* or even more simple, the one space jump could be learnt by separate classifiers.

While this thesis has focused on static concepts also non-static concepts could be considered for learning purposes. For instance classification of *sente* (forcing) and *gote* (non forcing) moves. This concept mostly applies to endgame situations. A *sente* move is basically so meaningful that it forces a response of the opponent, when this response is not absolutely necessary then it is a *gote* move. This does not mean that a *sente* is the best move as there still could be another bigger move. Another interesting classifier to learn would be whether stones can be captured or not. Many times, positions in the game occur in which a stone can be captured but from which is abstained as a rescue attempt from the opponent would be leading to damaging consequences.

The ideas where learning in the game can be applied on very specific topics are seemingly endless. The art of successful experimenting therefore lies in creating well defined expert data in combination with a board representation which then can be used successfully for the learning task.

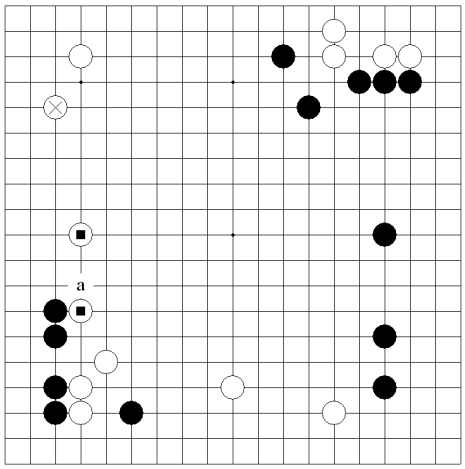
7.2 Making learning more efficient

Entering connectivity data is a difficult task for an expert, not only because the expert is required to think about for him/her appearing trivial issues but because it requires also a lot of "mousework". Now that first steps have already been made and a 69% correct classifier has been developed, one could think of creating an interactive environment where the program would propose the category of a connection not yet recorded in the training set. The selection of such a connection could be done by certain hard coded parameters or where high uncertainty of the classifier is apparent. A known approach to explore the feature space using expert interaction is active learning (see e.g. by Verbeek and Vlassis [30]). The expert can then simply agree with the opinion of the program by hitting enter or can correct the classification by hitting a number from 1-5. The connectivity entered can then be stored in a new training example and used for the learning of an improved model, which could be evaluated at the end of a tutoring session. This kind of approach would be greatly improved if sequential learning could be implemented alongside with the interactive environment. In case improved models and representations are developed the current model can easily be abandoned as the generated reliable expert data remains untouched.

7.3 Impact on Go programs

Pattern recognition always is paired with analytic ability, if a program possesses both skills good results can be expected. Until now the pattern recognition process was usually conducted by comparison with generated hand made patterns. The approach presented in this thesis is semi automatic and allows for the exchange between expert and program knowledge in a straightforward way. The connectivity classifier could be a significant asset for developing goals for a Go program, such as were to conduct local analysis. For instance, if the classifier classifies an opponent's connection to be conditional, then a local search of separation possibilities could identify the underlying reason for the recognition of the pattern and might seek action in that area when the timing and conditions are right. This way analytic computing power can be channelled into the right directions requiring the program's attention only when it is necessary.

Take the following example in figure 7.1. The connection between the two marked stones is considerable and the black player would restrain from cutting unless he would have a good reason. By labeling the connection conditionally connected like an expert would do, a program would keep in mind during the course of the game that separating the stones is a possibility.



(a) The marked white stones are conditionally connected. By playing at 'a' the stones can be connected or separated.

Figure 7.1: Example of how the classifier can be used

Appendix A

Used move and outcome types

In this appendix a list of move and outcome types used for labelling the *tesuji* problems using (commonly used) Japanese Go terminology is given. The underlying thought to present these lists here is to show readers, who are not very familiar with the game of Go, the richness of used go terminology and to show the advanced Go playing readers what kind of aspects have been dealt with during this research. The move type list has been presented without translations, because the expressions used are only applicable for the game of Go and describe the shape a move has relative to other stones. The outcome type list has been translated into English to demonstrate what kind of concepts a player is dealing with during a game. Sometimes two similar types are combined into a single one, which is indicated with a dash.

Outcome type	Description
arasu	to destroy territory
ikiru	to live with a group
ijimeru	to tease or torture a group
ukeru	to respond indirectly
utsutekaeshi	to capture with a snapback
eguru	to drill into territory
oiotoshi	create shortage of liberties of adjacent chains
osamaru	to solidify a group, to settle
kata o kimeru	to decide, to force a shape,
kata o kuzusu	to destroy the opponent's shape

Figure A.1: Outcome type list first part

Outcome type	Description
katameru	to make one's own shape stronger
kikaru	to take advantage of an opponent's weakness
ko	ko
korosu	to kill a group
sabaku	to place flexible stones
shicho - yurumu shicho	ladder, loose ladder
shinogu	to endure, to save seemingly dead stones
shibori o fusegu	to defend against a squeeze
shiboru	to squeeze
shimetsukeru	sacrifice a stone/stones to solidify
shinsuisuru	to escape from an enclosed position
suteishi	to sacrifice a stone, group of stones
seikei	connecting stones loosely
setsudan	to separate two groups (amputate)
semeai	liberty race
semeru	to attack a group
seriai	close and tight combat
sente o toru	take initiative / sente
dashutsu	to escape
dametsumari ni suru	creating a shortage of liberties
te o tsumeru	remove liberties from the inside
togameru	punish a weakness of the shape
toru	capture stones
hangeki	counter attack
fusa	enclose opponent's stones
fusegu	to close territory
fuyasu	increase territory
fundan	to separate groups
herasu	reduce territory
mamoru	to protect territory
yaburu	to capture a chain from a group
yosumi	to test the opponent's reaction
ryonirami	double attack
renzoku	to connect two groups
watari	to connect from underneath

Figure A.2: Outcome type list second part

Move type	
ate - atekaeshi	deru - tsukidashi
atekomi	dekiri
uetstuke - zutsuke - hanazuke	tobi - hazamatobi - chikiritobi
oki	tobitsuke
osae	narabi
oshi	nidanbane - nidanosae
kake	ninoichi
kaketsugi	nuki
kado - katatzuki	nozoki
kiri - kirichigai - chokiri - ishinoshita	nobi
guzumi	hai
keima - ogeima	hasamitsuke
kosumi	hane
kosumitsuke	hanekomi
sagari	hanedashi
shitatsuke	hiki
tsugi	fukurami
tsukiatari - butsukari - sashikomi	hekomi
tsuke	hourikomi
tsukekiri	mage - magari
tsukekoshi	yokutsuke - haratsuke
	warikomi

Figure A.3: Move type list

References

[1] Mark Boon, *A Pattern Matcher for Goliath*, Computer Go Nr. 13 pages 12-23, December 1989

[2] Mark Boon, Tesuji Software Go Library downloadable at: http://sourceforge.net/projects/tesujigolibrary/

[3] Bruno Bouzy, *The INDIGO program*, In Proceedings of the 2nd Game Programming Workshop in Japan, Kanagawa 1995

[4] Bruno Bouzy and Guillaume Chaslot, *Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 Go* In G.Kendall and Simon Lucas, editors, IEEE 2005 Symposium on Computational Intelligence in Games, Colchester, UK, pages 176-181, 2005

[5] B. Boser, I. Guyon, V. Vapnik, *A training algorithm for optimal margin classifiers*, In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pages 144152. ACM Press, 1992

[6] Tristan Cazenave and Bernard Helmstetter, *Search for transitive connections*, Information Sciences, Volume 175, Issue 4 , pages 284-295, 15 November 2005

[7] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*, 2001. Software available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm

[8] Fredrik A. Dahl, *Honte, a Go-playing program using neural nets* In Fürnkranz and Kubat Workshop Notes: Machine Learning in Game Playing. 16th International Conference on Machine Learning (ICML-99), Bled, Slovenia, 1999

[9] Markus Enzenberger, GoGui downloadable at
http://sourceforge.net/projects/gogui/

[10] Markus Enzenberger *Evaluation in Go by a neural network using soft segmentation*, In 10th Advances in Computer Games conference, pages 97-108, 2003

[11] David Fotland, *"Knowledge representation in The Many Faces of Go"*, 1993, Available by Internet

[12] GNU Go downloadable at:
http://www.gnu.org/software/gnugo/gnugo.html

[13] Thore Graepel, Mike Goutrie, Marco Krüger, and Ralf Herbrich, *"Learning on graphs in the game of Go"*, International Conference on Artificial Neural Networks , Vienna, Austria, 2001

[14] Frank de Groot, *Moyogo*, http://www.moyogo.com

[15] Arno Hollosi, *SGF file format standard and definitions*,
http://www.red-bean.com/sgf/

[16] Arno Hollosi, *XGF - An XML Game Format*,
http://www.red-bean.com/sgf/xml

[17] Arno Hollosi, *Sensei's Library - File formats*
http://senseis.xmp.net/?FileFormat

[18] www.kernel-machines.org

[19] A.Kishimoto and M.Müller, *Search versus knowledge for solving life and death problems in Go*, In Twentieth National Conference on Artificial Intelligence (AAAI-05), pages 1374-1379, 2005

[20] Byung-Doo Lee, *Life-and-death problem solver in go.* Technical Report CITR-TR-145, University of Auckland, 2004

[21] Tapani Raiko, *Nonlinear relational Markov networks with an application to the game of Go*, 15th International Conference on Artificial Neural Networks, ICANN 2005, pages 989-996, 2005

[22] Liva Ralaivola, Lin Wu, Pierre Baldi, *SVM and Pattern-Enriched Common Fate Graphs for the Game of Go*, ESANN 2005: pages 485-490

[23] Jan Ramon and Hendrik Blockeel, *A survey of the application of machine learning to the game of Go*, In Hahn and Sang-Dae, editors, First International Conference on Baduk, pages 1-10. Myong-ji University, Korea, May 2001

[24] W.S. Sarle, Part 2 of Neural Network FAQ, periodic posting to the Usenet newsgroup comp.ai.neural-nets, 1997

[25] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski *Temporal difference learning of position evaluation in the game of Go*, In Advances in Neural Information Processing 6. Morgan Kaufmann, 1994

[26] David Stern, Thore Graepel, and David J.C. MacKay, *Modelling uncertainty in the game of Go*, Advances in Neural Information Processing Systems 17, 2004

[27] David Stern, Ralf Herbrich, and Thore Graepel, *"Bayesian pattern ranking for move prediction in the game of Go"*, International Conference on Machine Learning (ICML-2006), 2006

[28] Rich S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning 3: 9-44, 1988

[29] Gerald Tesauro, *Temporal difference learning and TD-Gammon*, Communications of the ACM, Volume 38, Issue 3, March 1995

[30] Jakob J. Verbeek and Nikos Vlassis, *Gaussian fields for semi-supervised regression and correspondence learning*, Pattern Recognition, 39(10):1864-1875, 2006

[31] Thomas Wolf, *The program GoTools and its computer-generated tsume go database*, Proceedings of the Game Programming Workshop in Japan'94, Hakone/Japan, pages 84-96, Oct. 1994