A Model of Acquisition and Refinement of Deductive Rules in the Game of Go

Takuya KOJIMA

General Systems Studies

Graduate Division of International and Interdisciplinary Studies

The University of Tokyo

February 1, 1995

Contents

1 Introduction **1**

1.1 Background of this study 1

1.2 Scopes and purposes of this study 4

1.2.1 Scopes of this study 4

1.2.2 Purposes of this study 6

1.3 Overview of this study and this thesis 7

2 Preliminaries **9**

2.1 Rules of Go 9

2.2 Explanation-Based Generalization(EBG) 10

3 System **12**

3.1 Overview of this system 12

3.2 Terminology 13

3.3 Knowledge Base: Rules 15

3.3.1 Basic rules 15

3.3.2 Forcing rules 15

3.4 Decision Maker: Decision making during a game 17

3.5 Rule acquisition 18

3.5.1 Terminology 19

3.5.2 Rule Acquisition Algorithm 19

3.5.3 An example of rule acquisition 23
3.5.4 Validity of Rule Acquisition Algorithm 27
3.5.5 Other results of rule acquisition 29
3.6 Rule Refinement 31
3.6.1 Rule Refinement Algorithm 33
3.6.2 Explanation of Rule Refinement Algorithm 34
3.6.3 Examples of Rule Refinement 36
3.6.4 Discussion on Rule Refinement Algorithm 38

4 Cognitive Experiments 40
4.1 Comparison of Rule Acquisition Algorithm with humans 40
4.1.1 Observation of humans – a preliminary experiment 40
4.1.2 A cognitive experiment 41
4.2 Comparison of Rule Refinement Algorithm with humans 48
4.2.1 Experiment 1 48
4.2.2 Experiment 2 49

5 Discussion 51
5.1 Comparison with other learning algorithms in adversary games 51
5.1.1 Numerical learning in adversary games 51
5.1.2 Other knowledge acquisition systems in adversary games 53
5.2 Other knowledge representation in Go 54
5.3 Comparison with original EBG 56
5.4 Future works 57

6 Conclusion 60

Acknowledgement 62

Bibliography 63

Appendix **68**

List of Figures

1.1 Background of this study. 5

2.1 Examples of liberties. 10

2.2 Examples of atari. 10

3.1 Overview of this system. 13

3.2 Two applications of forcing rules. 17

3.3 Overview of Rule Acquisition algorithm. 18

3.4 Problem 1. 23

3.5 Solution of Problem 1. 23

3.6 Board position on Move 5. $\delta(BC_4, s_5) = BC_5$. 23

3.7 Explanation of Move 5. Contents of Relevant Parts (RP_5). 23

3.8 An explanation tree for Move 5. 24

3.9 Board position on Move 4. $\delta(BC_3, s_4) = BC_4$. 25

3.10 Explanation of Move 4. Contents of Relevant Parts (RP_4). 25

3.11 Board position on Move 3. $\delta(BC_2, s_3) = BC_3$. 25

3.12 Pre-rule made in Step 10. 25

3.13 Pre-rule in Prolog. 26

3.14 Rule f2, which is acquired by generalizing pre-rule in Step 11. 26

3.15 A process of extracting relevant parts. 27

3.16 Board position on Move 1. $\delta(BC_0, s_1) = BC_1$. 27

3.17 Rule f3. 27

3.18 A process of rule acquisition. 28
3.19 A problem and its answer of geta. 30
3.20 An acquired rule of Geta. 30
3.21 A problem and its answer of ashida. 30
3.22 An acquired rule of Ashida. 30
3.23 A difference of a search tree between of non-adversary games and of adversary games. 31
3.24 An example of over-generalization. 32
3.25 Normal Rule Acquisition. 32
3.26 Rule Refinement by adding a restriction. 33
3.27 An example of refinement of an acquired rule. 37
3.28 A problem and its solution. 38
3.29 A rule acquired from Fig. 3.28. 38
3.30 A negative example for the rule in Fig. 3.29. 38
3.31 Relevant Parts on Move 2 (RP_2). 39
3.32 Relevant Parts on Move 1 (RP_1). 39
3.33 Rules acquired from Fig. 3.30. 39

4.1 The learning problem. 43
4.2 The model suggestion of the learning problem. 43
4.3 An example of $(CS, SM) = (\circ, \circ)$. 43
4.4 An example of $(CS, SM) = (\circ, \times)$. 43
4.5 An example of $(CS, SM) = (\times, \times)$. 43
4.6 The given problem in Experiment 1. 49
4.7 The given problem in Experiment 2. 49
4.8 The process given in Experiment 2. 49
4.9 The results of subjects of Experiment 2. 50

4.10 The results of Rule Acquisition Algorithm of the same problem as Experiment 2. 50

5.1 Examples of atari. 54

5.2 An example of positions which belong to the same category for some novices. 59

List of Tables

4.1 Results of Subject A. 45
4.2 Results of Subject B. 45
4.3 Results of Subject C. 46
4.4 Results of Subject D. 47
4.5 Results of Subject E. 47

6.1 Results of world computer Go competitions 68

Chapter 1

Introduction

1.1 Background of this study

Since the birth of AI, it is one of the goals of the researchers [7,37] to build the computer machine which beats the top human players. Through the struggle for building chess playing computers, many techniques are brought to AI and sometimes chess is regarded as *Drosophila*(fruit fly) of AI [22], which means that chess is a good testing ground for algorithms and data structures. Main contribution of chess to AI is to develop several search techniques, such as $\alpha - \beta$ algorithm, minimal window search, iterative deeping and hash table [19]. Now AI nearly reaches the goal; a chess playing computer system named Deep Thought (or Deep Blue, which is a newer version of Deep Thought) [10, 11] is now on a level with the top of human professionals. On August 31, 1994, another computer program named Genius II, which is a software for personal computers, beat a human strongest player, Gary Kasparov, in 30 minutes match by 1.5-0.5. Therefore it can be said that the role of chess in AI is almost completed.

Problem solving methods of both humans and machines, in general, can be classified into two main categories: by search and by inference. Solving a maze, for example, is a problem solving by search and expert systems solve problems by inference.

In the same way, there are two ways for building game playing programs as shown

in Fig. 1.1: one is the way based on search and another is that based on inference using knowledge. Chess programs have adopted the former way. A reason for the success of adopting the former way in chess is that the average branching factor in chess is about 35 and it is easy for the computers of today to search all the possible moves of both players. This approach is called *brute force search* and it has proved very effective in chess [2,11,12], checkers [34], Othello [17,28], backgammon [1] and many other games [18,43]. Therefore, chess as the old drosophila of AI was really a drosophila of problem solving by search or of brute force search.

Recently the game of Go has attracted considerable attention of AI researchers as the target next to chess. On Thirteenth International Joint Conference on Artificial Intelligence, which held in 1993, there was a panel named "Grand Challenge AI Applications Proposed in Japan", where building Grand Master Computer Go Systems is declared as one of the Grand Challenges in Japan [14]. John McCarthy and other researchers regard Go as a new drosophila for AI [20].

Although many computer Go systems have been built [5,9,26,33], their performances are on a level with those of human novices; at best they reach the level of 8 *kyu* [21]. Main reasons for their low performances are that Go has a much higher branching factor than chess, which is over 200, and that a brute force search is not very effective in finding proper moves in Go.

In place of brute force search, Go programs have adopted the latter approach (Fig. 1.1), or the *knowledge intensive approach*. Thus, Go is a drosophila of problem solving by inference or of knowledge intensive approach. Most computer programs use patterns of the board, which are designed by programmers. For example, Many Faces of Go [9], which took the second place in the 1994 computer Go competitions (See Appendix 6), has a pattern database which consists of about a thousand patterns [8].

In order to make programs stronger, much more patterns are required. In fact, human players are said to know much more than a thousand patterns. Since it is too hard

to implement such enormous patterns, it is necessary for computer programs to acquire patterns for themselves. No pattern acquisition algorithms, however, have been proposed in Go. Although there are some studies which attempt to acquire patterns in the game of Go-moku [23, 24], it is necessary to give the State-Update rules, which are hard to be given in Go. Therefore, a new pattern acquisition algorithm which is powerful enough to be applied to Go is needed to be developed.

It is, in general, hard to acquire perfect rules in adversary games, especially in Go, because most of the Go knowledge is situation-dependent. Thus refinement of acquired rules is indispensable in Go. None of the pattern acquisition systems in adversary games, however, have had refinement algorithms implemented. Therefore a new refinement algorithm should be proposed in order to build rule acquisition systems.

Since humans show much higher performances in playing Go than existing Go playing programs, a human-like method is considered effective. Only a small number of attempts, however, have so far been made on Go play in terms of cognitive studies [**?**, 29, 30, 42], which is true of other games. Furthermore, as for learning, no systems of adversary games have been compared with humans. Therefore a system whose behaviors can be compared with those of human players are needed to be built.

K. VanLehn says in [41] on page 553, as follows:

> No computer program yet exists that can start off as a novice in some knowledge-rich task domain and slowly acquire the knowledge needed for expert performance. Thus we lack even a computationally explicit account of the novice-expert transition, let alone one that compares well with the performance of human learners.

As VanLehn points out, it is a very challenging task to build a program which can start off as a novice in some knowledge-rich task domain and slowly acquire the knowledge needed for expert performance.

In addition, most problem solving systems using knowledge have the same weakness:

the hierarchical structure of knowledge is abnormal. Those who know about difficult things in a domain know also about easier things in the same domain. The more difficult the knowledge is, the less they have. It is often the case with most expert systems, however, that they know about difficult things but does not know about easier things in the same domain. In addition, since most knowledge is given, their knowledge has no ground to be based on. Expert systems do not know why the rules hold which they have. Most of the rules are given by human experts without explanation [4]. Therefore, the systems cannot change rules which proved to be false. Case-Based Reasoning(CBR) systems also have the same weakness.

The background of this study is summarized in Figure 1.1. In short, there are four problems in the previous studies on building Go learning systems. First, previously proposed pattern acquisition algorithms are not powerful enough to acquire patterns in Go. Secondly, although rule refinement is indispensable to learning systems in Go, no rule refinement algorithms have been proposed in adversary games. Thirdly, no learning systems in adversary games have been compared with humans, which will be necessary to build stronger computer programs and important in terms of cognitive studies. Finally, the hierarchical structure of most problem solving systems is not normal and most expert systems do not know the reason of rules to hold, which is necessary for improving the rules.

1.2 Scopes and purposes of this study

1.2.1 Scopes of this study

In this study, players are classified as follows in this study:

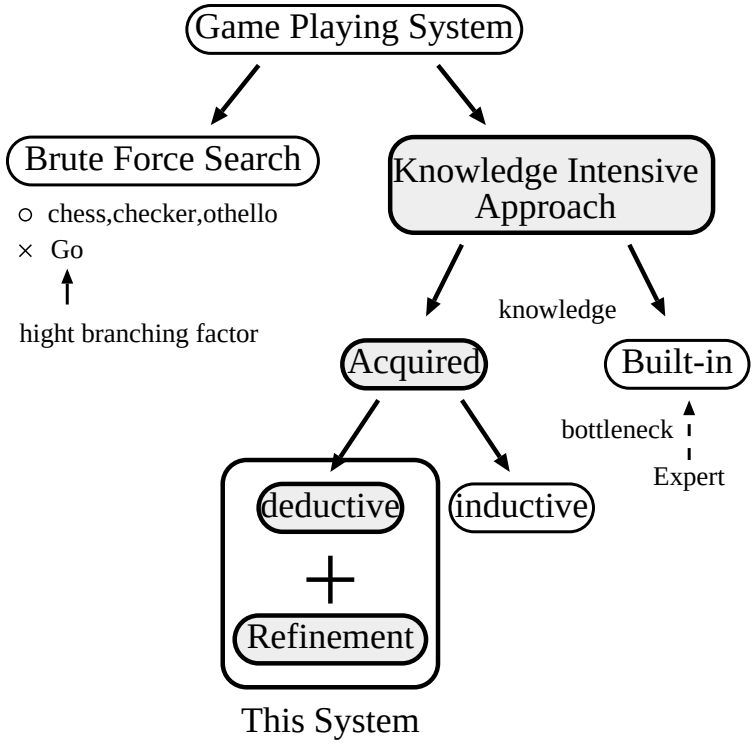

Figure 1.1: Background of this study.

- Player
  - Expert
  - Intermediate
  - Beginner
    - Novice (20 ∼ 10 kyu)
    - Pre-novice (start ∼ 20 kyu)
    - The unexperienced (just started)

The ultimate goal is to build a system as VanLehn mentions, which can start off as a novice in some knowledge-rich task domain and slowly acquire the knowledge needed for expert performance. Toward this goal, learning in beginners are mainly concerned in this study.

Knowledge in Go which novices should know can be classified as follows in this study:

- Knowledge for novices
  - Joseki
  - Life and Death
  - Connectivity
  - Way to capture stones

The upper knowledge is based on the lower one.

Aiming the system VanLehn mentions, acquisition of rules concerning the way to capture stones, which is the lowest knowledge, is concerned in this study.

1.2.2 Purposes of this study

It is the purpose of this study to construct a system that partly solves the problems of the previous studies, which are mentioned in Section 1.1.

In the first place, a rule acquisition algorithm in Go will be proposed in this study, which partly solves the first problem. The target problems are stone-capturing problems, the basic knowledge in Go, which is a solution to the fourth problem.

In the second place, a rule refinement algorithm in Go is also proposed, which has never been proposed in adversary games. This partly solves the second problem.

Finally, these models are compared with humans, which has rarely been attempted not only in adversary games but also in other fields. This is a solution to the third problem.

1.3 Overview of this study and this thesis

In the first place, two novices are selected as the subjects and their protocols while playing Go and while solving Go problems are recorded. The analysis of the protocols indicates that human novices may acquire rules deductively. In addition, it indicates that they may refine the acquired rules when they meet the negative examples against the acquired rules.

Next, both a model of rule acquisition and a model of rule refinement are proposed. Since human beginners may acquire rules deductively, the models partly use Explanation-Based Generalization (EBG), which is a typical model of deductive learning.

Finally, the models are compared with human beginners by making experiments. In the experiment for comparing the rule acquisition model with humans, pre-novices and the unexperienced persons are selected as subjects in order to see whether those who are under the level of novices are capable of acquiring rules deductively. In the experiment for comparing the rule refinement model, novices are chosen as subjects.

In Chapter 2, the preliminaries will be explained; rules of Go and the basic concepts of Explanation-Based Generalization (EBG) will be explained.

The details of the system will be explained in Chapter 3. The proposed system will be explained according to its modules; Knowledge Base will be explained in Sections 3.3, Decision Maker in 3.4, Rule Acquisition Module in 3.5, Rule Acquisition Module in 3.6.

In Chapter 4, the cognitive experiments will be explained; in Section 4.1 Rule Acquisition Algorithm are compared with humans, and in Section 4.2 Rule Refinement Algorithm are compared with humans.

In Chapter 5, this system is discussed. In Section 5.1, this system is compared with other learning algorithms. In Section 5.2, relations between the different knowledge representation in Go and the proposed knowledge representation are discussed. In Section 5.3, this system is compared with original EBG, which is partly used in this system. In Section 5.4, Future works are explained.

Chapter 6 concludes this thesis.

Chapter 2

Preliminaries

In this chapter, rules of Go and Explanation Based Generalization (EBG) are explained.

2.1 Rules of Go

In this section, some rules of Go are explained, which are used in the following sections.

Go is a board game between two players, Black and White. The players take turns, and on each turn each player puts a stone of his/her color(Black or White) on a board. Stones are placed on intersections of vertical and horizontal lines on the board, including edges and corners. The board used in Go is usually 19×19, and smaller boards such as 9×9, 11×11 or 13×13 are used by beginners in order to reduce the difficulties.

Stones of the same color adjacent to each other horizontally and vertically form a **string**. A vacant intersection adjacent to a stone or a string is called a **liberty**. A stone or a string must have at least one liberty. When a stone or a string has no liberty, it is removed from the board, called **captured**. When a stone or a string has only one liberty, it is called in **atari**.

Take Figure 2.1 as an example. In the right figure, stones marked with the same letter **a**,**b** and **c** belong to the same string. Liberties of a stone or a string are marked by **L**. In Figure 2.2, all White stones are in atari and the places marked **a** are the last liberties for

each stone or string.

Figure 2.1: Examples of liberties. **L** stands for liberties of White stones.

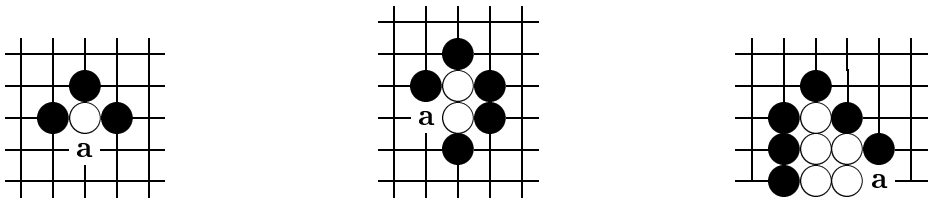

Figure 2.2: Examples of atari. All the White stones are in atari and **a** is the last liberties.

A player can pass at any time, and a game is over when both players continuously pass. The object of the game is to surround territory and capture opponent's stones. An intersection a player surrounds and a stone a player captures count as a point, and the player with the more points is the winner.

While the ordinal rule of Go is as mentioned above, the object in this thesis is to capture opponent's stones. The player who captures opponent's stones earlier wins.

2.2 Explanation-Based Generalization(EBG)

In this system, Explanation-Based Generalization(EBG) [25] is used. In the program Prolog EBG [13] is used. EBG is explained in this section.

EBG has the following four inputs: [25]

goal concept A concept definition describing the concept to be learned.

training example An example for the goal concept.

domain theory A set of rules and facts to be used in explaining how the training example is an example of the goal concept.

operationarity criterion A predicate over concept definitions, specifying the form in which the learned concept definition must be expressed.

An aim of EBG is to put the goal concept in another way, the sufficient condition of the goal concept.

EBG consists of two steps – explanation and generalization [25]. In this thesis, only the explanation step is used. In the explanation step, an explanation tree is made to explain why the training example is an example of the goal concept; the tree is made to explain why the training example suffices the goal concept.

An important thing in this explanation step is that by making the tree, only the relevant features are selected among many facts, which is used in Rule Acquisition Algorithm (Section 3.5).

Chapter 3

System

This system plays Go. When the system loses against someone, it learns some rules. When it realizes that some rules are inaccurate, it refines these rules. This system is implemented by Prolog.

As mentioned in Section 2.1, the object of this system is to capture opponent's stones earlier than the opponent. Thus, to *win* and to *lose* are defined as follows:

to win To capture opponent's stones.

to lose To be captured stones by opponents.

In this chapter, the configurations of this system and the algorithms to acquire and refine rules are explained.

3.1 Overview of this system

This system consists of four modules: Knowledge Base (K.B.), Decision Maker, Rule Acquisition Module and Rule Refinement Module. These modules are explained in detail in Section 3.3, 3.4, 3.5 and 3.6, respectively.

The system plays Go against someone. During a game Decision Maker searches appropriate rules in K.B., and applies rules if it finds them; otherwise it puts stones on

a vacant place randomly. After the game, if the system loses, Rule Acquisition Module acquires some rules and add them to K.B. When the system cannot win as the result of applying rules for attack, Rule Refinement Module refines these rules. In this refinement, it extracts these bad rules from K.B. and refines them and add new refined rules to K.B. See Figure 3.1.

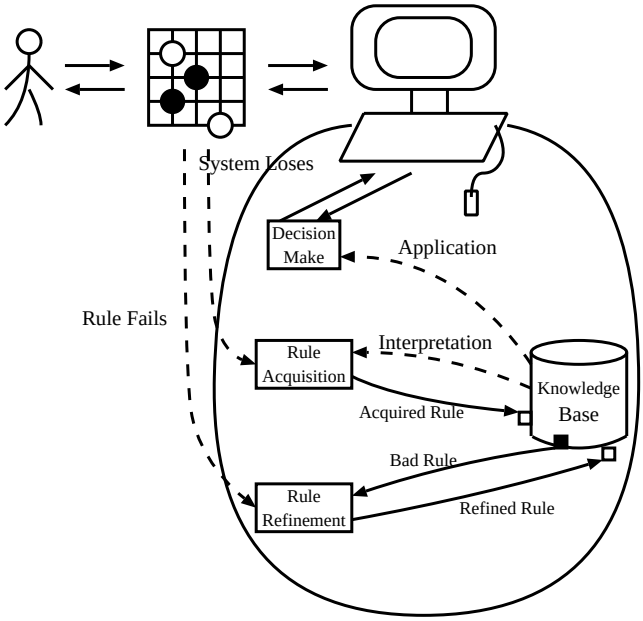

Figure 3.1: Overview of this system.

3.2 Terminology

Some terms used in this thesis are introduced.

last move M is the number of moves in a game (the last move number).

color In Go, there are two colors, Black(B) and White(W). $C = \{B, W\}$.

board size A board size BS is the number of horizontal or vertical lines on the board. It is a constant variable through a game. As mentioned in Section 2.1, in ordinary Go a board is 19×19, $BS = 19$. Sometimes it is set to 9, 11 or 13.

board configuration BC is a set of board configurations. $BC_m(0 \leq m \leq M)(\subseteq BC)$ is a board configuration after Move m. BC_0 is the initial board configuration. Thus $BC = \{BC_i | 0 \leq i \leq M\}$. BC_m is described as a 4-tuple $\langle t_m, S_m, V_m, E \rangle$ where:

turn $t_m(\in C)$ represents the color of turn after move m.

stone S is a set of stones and S_m $(\subseteq S)$ is a set of those placed from the initial position to Move m. S_0 is a set of stones which are placed in the initial position. $S_m = S_0 \cup \{s_i | 1 \leq i \leq m\}$, where s_i is a stone which is put on Move i. A stone s_i is a 2-tuple:

$$s_i = \langle [x_i, y_i], c_i \rangle \ (1 \leq x_i \leq BS,\ 1 \leq y_i \leq BS,\ c_i \in C),$$

where $[x_i, y_i]$ is a coordinate of the place on which the stone is put.

vacant V is a set of coordinates of vacant places and $V_m(\subseteq V)$ is that after Move m. Thus the definition of V_m is as follows:

$$V_m \stackrel{\text{def}}{=} \{v_{x,y} | 1 \leq x \leq BS,\ 1 \leq y \leq BS,\ \langle [x, y], c \rangle \notin S_m (c \in C)\}$$

edge E is a set of coordinates of edges. Thus the definition of E is as follows:

$$E \stackrel{\text{def}}{=} \{e_{x,y} \mid (0 \leq x \leq BS + 1) \wedge (0 \leq y \leq BS + 1) \wedge (x = 0 \vee x = BS + 1 \vee y = 0 \vee y = BS + 1)\}$$

move δ is a function which represents a move. $\delta(BC_{m-1}, s_m) = BC_m$, for example, represents that in a board configuration after Move $m - 1$ (BC_{m-1}), Move m yields BC_m by putting Stone s_m.

don't care * mark represents 'don't care'.

3.3 Knowledge Base: Rules

Knowledge Base has two kinds of rules: *basic rules* and *forcing rules*.

3.3.1 Basic rules

Basic rules (BR) are definitions of basic terms in Go. The following terms are defined:

capture If stones have no liberty, they are captured; stones which are completely surrounded by opponent's stones are captured.

atari If stones have only one liberty, they are in atari; if stones are captured in a move, they are in atari. An algorithm which judges whether a stone is in atari or not is shown below. Atari is described as $\text{atari}_{s_i}^{x,y}$, which represents that a stone, s_i, is in atari and its last liberty is $[x, y]$.

r_color A function which returns the reverse color of an input. For example, $\text{r_color}(B) = W$.

Equations such as $x = x_1 + 1$ are also basic rules.

Atari is defined in this system as follows:

Focus on a stone, s_i. Collect all the adjacent places of s_i. If these places are occupied by a stone, s_j, with the same color as s_i, then collect all the adjacent places of s_j. These collecting processes should continue recursively. The stone s_i is in atari, iff the number of the collected spaces is one which are neither edge nor occupied by a stone with the different color from s_i.

3.3.2 Forcing rules

Forcing rules are defined as follows: a player always wins if he/she applies forcing rules. They are represented as follows:

$$rule_n^{for} : BC \cup BR \rightarrow \delta(s, BC)\ (s = \langle [x, y], c \rangle),$$

where n is the rule number.

An example of forcing rules is shown below:

$$\text{r_color}(t) = t',\ s_k = \langle [*,*], t' \rangle,\ \text{atari}_{s_k}^{x,y} \ \rightarrow\ \delta(s_m, BC)\ (s_m = \langle [x,y], t \rangle) \qquad \text{(f1)}$$

which means as follows:

if opponent's stones are in atari, **then** put a stone on their last liberty.

This rule is defined in Prolog as follows:

```
rule(for,1,Color,Put_here) :-
   r_color(Color,Rcolor),
   stone(Stone_no,_,Rcolor),
   atari(Stone_no,Put_here).
```

There are two ways to apply forcing rules: one is to apply them for attack and the other to apply for defense. When a player applies a forcing rule for the sake of his/her own attack, it is called an *application for attack*. When he/she recognizes that his/her opponent can apply a forcing rule in the next turn, a player may make a move to escape opponent's application for attack; that is called an *application for defense*. Take Figure 3.2 for example. In the figure, all the White stones are in atari and all the places marked **a** in the figure are the last liberty of White stones. Thus, when Black puts a stone on **a**, the move is an application of Rule f1 for attack; when White puts a stone on **a**, the move is an application of Rule f1 for defense.

In the initial state, the only given rule to the system is Rule f1 shown above. Other forcing rules are acquired by Rule Acquisition Module.

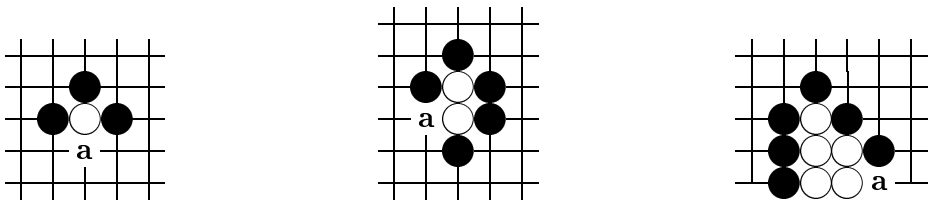

Figure 3.2: Two applications of forcing rules. When *Black* puts a stone on **a**, it is called application *for attack*; when *White* puts on **a**, it is called *for defense*.

3.4 Decision Maker: Decision making during a game

Decision Maker uses only forcing rules. If there is no forcing rules to be applied, a stone is put randomly in a vacant place. Decision Maker makes a move according to the following algorithm:

Algorithm 3.1 Decision Making

1 **if** able to apply Rule f1 for attack
  then apply Rule f1 for attack

2 **if** opponent is able to apply Rule f1 for attack
  then apply Rule f1 for defense

3 **if** able to apply a forcing rule other than Rule f1
  then apply the rule for attack

4 **if** opponent is able to apply a forcing rule other than Rule f1
  then apply the rule for defense

5 Choose a place to put a stone randomly in a vacant place.

In general, rules are applied for attack (Step 1, 3) earlier than for defense. The naïve reason is that in your turn you should attack before defense.

However, Rule f1 is an exception. One should apply Rule f1 for defense before attack applying other rules because rules other than Rule f1 take generally more than 1 move to win [1]; when one applies Rule f1 for defense the opponent will apply the rule in the next

[1] In this paper, to win is to capture stones as mentioned above.

turn and capture one's stones, so one has no time to take opponent's stones.

Although only Rule f1 is treated as an exception in this thesis, it is not enough. In many cases the order of applying rules should be considered. Details are in Section (Discussion).

3.5 Rule acquisition

In this section, an algorithm for rule acquisition is proposed, which consists of three steps. The first step chooses good moves to be learned from a sequence of moves in a game by interpreting moves using forcing rules. The second step extracts relevant parts from the board position where the learning moves are made by using EBG on each move. The last step generalizes the position and the move by changing their coordinates from constants to variables. See Figure 3.3.

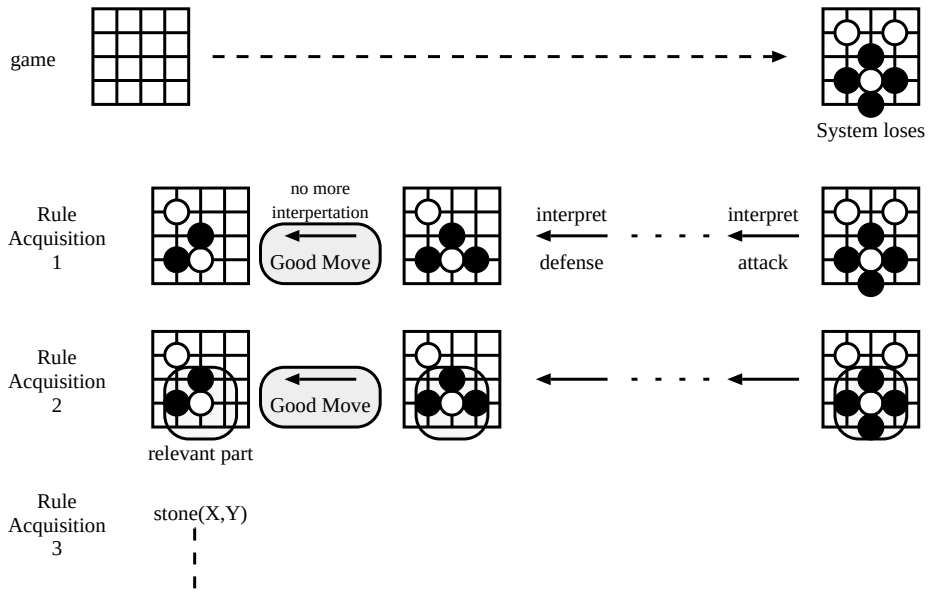

Figure 3.3: Overview of Rule Acquisition algorithm.

3.5.1 Terminology

Some terminologies used below are introduced here.

interpretation i_m is an interpretation of Move m and a list `[`m`,`r`,`ad`,`wl`]` where:

m**:** represents that this interpretation is that of Move m.

r**:** denotes the kind and the number of a rule of which the move is interpreted as application of a rule. It is usually 'f'(forcing rule)+i(the number of a forcing rule).

ad**:** $ad \in \{$`a`,`d`$\}$, where `a` represents application for attack and `d` that for defense.

wl**:** $wl \in \{$`w`,`l`$\}$, where `w` represents that the move is interpreted as application of the rule by a winner and `l` by a loser.

Interpretation Sequence IS is a list of interpretations of moves.

$$IS = [i_M, i_{M-1}, \ldots, i_m](1 \leq m \leq M).$$

Note that interpretations are lined in reverse order of the moves in the game.

relevant parts RP ($\subset BC$) is a set of relevant parts of a board configuration. Thus $RP = S \cup V \cup E$. RP_m is a set of parts relevant to Move m.

3.5.2 Rule Acquisition Algorithm

Rule Acquisition Algorithm is shown below:

Algorithm 3.2 Rule Acquisition

$m \leftarrow M$, $IS \leftarrow \phi$, $RP \leftarrow \phi$, j =the number of forcing rules

Loop 1

Loop 2

1 $i_m \leftarrow$ Interpret$(\delta(BC_{m-1}, s_m) = BC_m)$

2 **if** i_m=ϕ **then exit Loop 2**

3 **if** i_m=`[m,f*,a,l]` **then Refinement**

4 $IS \leftarrow IS + i_m$

5 $RP_m \leftarrow$ Explain(i_m)

6 $RP \leftarrow RP \cup RP_m - s_m$

7 $m \leftarrow m - 1$

end Loop 2

8 **if** $i_{m+1} \neq$ `[`$m+1$`,f*,d,l]` **then exit Loop 1**

9 $RP \leftarrow RP - s_m(= \langle [x, y], c \rangle) + v_{x,y}(\in V)$

10 pre-rule $\leftarrow \langle RP \subset BC_i \rightarrow s_m \rangle$

11 $rule^{for}_{j+1} \leftarrow$ Generalize(pre-rule)

12 KB $\leftarrow$ KB + $rule^{for}_{j+1}$

end Loop 1

The algorithm investigates moves from the last move to the initial one, move by move. Loop 2 continues while moves which are made by both players can be interpreted as application of forcing rules the system possesses. In Loop 2, parts of board positions relevant to the interpretation(RP) are extracted. When Loop 2 is over, the interpretation should be a defense move of a loser. Thus the move just before Loop 2 is over should be interpreted as application for attack by a winner. Therefore a rule is acquired by generalizing the board position and the move. Loop 1 continues to acquire rules after adding the rule.

Procedures used in Step 1, 5 and 11 (Interpret, Explain and Generalize, respectively) are explained below. Explanation of the procedure, Refinement, in Step 3 will be given in Section 3.6.

Step 1: Interpretation of a move

In this step, the system interprets moves of both players; the system tries to understand moves as the application of a forcing rule. If Move m can be understood as application

of a forcing rule, the output is the interpretation,i_m; otherwise, the output is ϕ.

Step 5: Explanation of the reason for an interpretation

In Step 5, parts of a board position (RP) are extracted from a $BS \times BS$ board which are relevant to the interpretation in Step 1(i_m).

Explanation Based Generalization (EBG), which is explained in Section 2.2 is used to extract the relevant parts. As explained before, EBG has four inputs: the goal concept, the training example, the domain theory and the operationarity criterion. In Rule Acquisition Algorithm, the four inputs are as follows:

goal concept the definition of a rule in the interpretation

training example Move $m(s_m)$, the board position before the move(BC_{m-1}), and the interpretation of the move (i_m).

domain theory forcing rules and basic rules in Knowledge Base.

operationarity criterion to paraphrase the goal concept by using basic configurations (BC) and basic rules (BR).

In Rule Acquisition Algorithm, an explanation tree is made to explain that the training example is really a positive example of the goal concept. An important thing in this explanation step is that by making the tree, among many facts only the relevant features are selected. Thus the main idea of Step 5 is that relevant parts will be selected from all board configurations of $BS \times BS$ board by making an explanation tree which explains why the interpretation is made.

Step 11: generalization of pre-rule

Since coordinates and colors in pre-rules $\langle RP \rightarrow s_m \rangle$, which are made by Step 10, are constant, they are changed into variables in order to expand the range of the application of

the acquired rule; the coordinates and the colors of parts are replaced with corresponding variables keeping the relative positions and colors.

Take the following example:

$$\text{pre_rule}: \ s_2(=\langle[5,1],W\rangle), e_{5,0}, v_{4,1} \rightarrow s_*(=\langle[4,1],B\rangle),$$

which means that if there is a stone, s_2 (whose color is White and which is on $[5,1]$), $[5,0]$ is an edge and $[4,1]$ is a vacant place, then put a stone, s_* (whose color is Black and which is on $[4,1]$).

In the first place, colors are changed into variables. Color B is replaced by a variable, c_1. As W is a reverse color of B, a basic rule, r_color is used; B is replaced by a variable, c_2 and $\text{r_color}(c_1) = c_2$.

Next, a coordinate, [4,1] is replaced by a list of variables,$[x,y]$. Coordinates in the IF part, [5,1],[5,0] and [4,1] are replaced by lists of variables, $[x_1,y_1],[x_2,0]$ [2] and $[x_3,y_3]$, respectively. In order to keep the relative relations between [4,1] and other coordinates of IF part, the following equations are added:

$$\begin{aligned} x &\overset{\text{unify}}{=} x_1 - 1, \quad y \overset{\text{unify}}{=} y_1 - 0, \\ x &\overset{\text{unify}}{=} x_2 - 1, \\ x &\overset{\text{unify}}{=} x_3 - 0, \quad y \overset{\text{unify}}{=} y_3 - 2 \end{aligned}$$

Therefore the generalized rule is as follows:

$$\begin{aligned} &\text{r_color}(c_1) = c_2, \\ s_*(&\overset{\text{unify}}{=} \langle[x_1,y_1],c_2\rangle), \quad x \overset{\text{unify}}{=} x_1 - 1, y \overset{\text{unify}}{=} y_1 - 0, \end{aligned}$$

[2] [5,0] is an edge because y-coordinate is 0, thus y-coordinate is not replaced by a variable and remains 0.

$$e_{x_2,0}, \qquad x \overset{\text{unify}}{=} x_2 - 1,$$

$$v_{x_3,y_3}, \quad x \overset{\text{unify}}{=} x_3 - 0, y \overset{\text{unify}}{=} y_3 - 2 \quad \rightarrow s_*(\overset{\text{unify}}{=} \langle [x,y], c_1 \rangle)$$

3.5.3 An example of rule acquisition

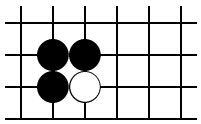

Figure 3.4: Problem 1.

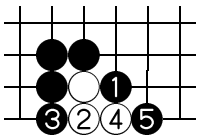

Figure 3.5: Solution of Problem 1.

To explain Rule Acquisition Algorithm concretely, take Problem 1 (The problem is shown in Fig. 3.4 and its solution in Fig. 3.5.) as an example.

Since the number of the last move of the solution is 5, 5 is substituted into m. Interpretation Sequence (IS) and Relevant Parts (RP) are set to an empty set. As the system has only one forcing rule, Rule f1, j is set to 1.

Step 1 interprets ❺. In Figure 3.6, the position before Move 5 (BC_4), all White stones are in atari and Black to move. Thus ❺ is interpreted as the application of Rule f1 for attack by a winner, which is denoted by `[5,f1,a,w]`; therefore $i_m =$ `[5,f1,a,w]`.

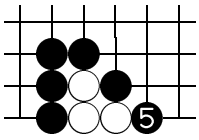

Figure 3.6: Board position on Move 5. $\delta(BC_4, s_5) = BC_5$.

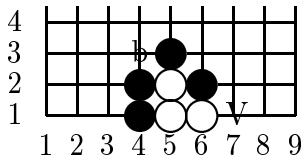

Figure 3.7: Explanation of Move 5. Contents of Relevant Parts (RP_5).

In Step 2, as i_5 exists, Loop 2 continues.

Step 4 adds this interpretation (i_5) to Interpretation Sequence (IS) to make it `[[5,f1,a,w]]`.

Step 5 extracts parts relevant to the interpretation `[5,f1,a,w]` from Fig. 3.6; EBG explains the reason why the system interprets ❺ as the application of Rule f1 for attack by the winner. As the consequence, the explanation tree as Fig. 3.8 is made.

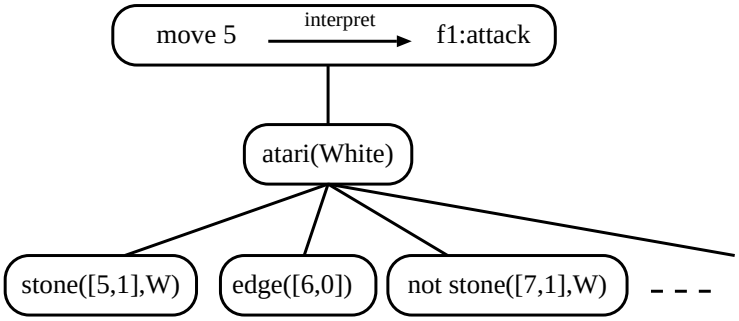

Figure 3.8: An explanation tree for Move 5.

The reason for the interpretation is that three White stones are in atari; four Black stones just adjacent to the White ones make White ones in atari. Thus the output of EBG is as Fig. 3.7, where the place denoted **b** is vacant because the Black stone on place **b** does not help Black to make White in atari. At the same time, the place where ❺ was put in Fig. 3.6 is marked as **V** in Fig. 3.7, which means that the place should be vacant to apply the rule. That is because the place is the last liberty for White and should be vacant in order that the White stones are in atari.

Step 6 adds the relevant parts (RP_5) as shown in Fig. 3.7 to Relevant Parts (RP). Since ❺ (s_5) is not in RP, s_5 is not retracted.

Step 7 subtracts 1 from m and makes m 4. Now the system is focused on Move 4.

Loop 2 continues and the system goes to Step 1.

In Step 1, as ④ is interpreted as the application of Rule f1 for defense by a loser (the system), the output (i_4) is `[4,f1,d,l]`.

In Step 2, as i_4 exists, Loop 2 continues.

Step 4 adds the interpretation (i_4) to Interpretation Sequence (IS) to make IS `[[5,f1,a,w], [4,f1,d,l]]`.

The output of Step 5 is shown in Fig. 3.10.

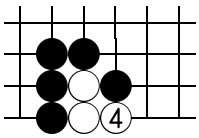

Figure 3.9: Board position on Move 4. $\delta(BC_3, s_4) = BC_4$.

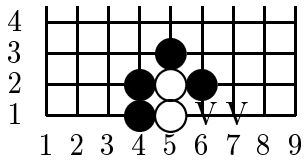

Figure 3.10: Explanation of Move 4. Contents of Relevant Parts (RP_4).

In Step 6, as the relevant parts acquired in Step 5 (RP_4) which are not in Relevant Parts (RP) are a vacant place, $v_{6,1}$, the vacant place is added to RP. Since s_4 no longer exists, s_4 is extracted from RP. As a consequence, $RP = RP_4$.

As Step 7 subtracts 1 from m, m is 3 and the move focused is Move 3.

Loop 2 continues and goes to Step 1.

In Step 1, i_3 is empty (ϕ) because ❸ cannot be interpreted by the only rule the system possesses (Rule f1).

Thus, in Step 2, Loop 2 is over.

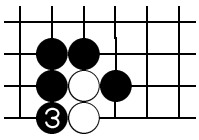

Figure 3.11: Board position on Move 3. $\delta(BC_2, s_3) = BC_3$.

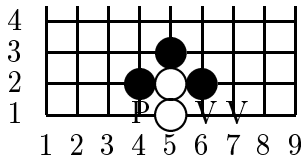

Figure 3.12: Pre-rule made in Step 10. **P** represents the place to put a stone.

The move and the position when Loop 2 is over are the move and the position to be learned, which are called the *learning move* and the *learning position*, respectively. Thus Move 3 is the learning move and the position just before Move 3 (BC_3) is the learning position.

In Step 8, since the interpretation before the focused move, i_4 is `[4,f1,d,l]` and matches with `[`$m+1$`,f*,d,l]`, Loop 1 continues.

In Step 9, in Fig. 3.11, ❸ (s_3) is retracted from Relevant Parts (RP). The place where

s_3 was put is vacant, thus $v_{4,1}$ is added to RP.

Step 10 makes a pre-rule:

if BC_i contains RP, **then** put a stone on the same place as s_m.

The rule is shown in Fig. 3.12, where **P** represents the place on which a stone should be put. The rule acquired in this step is described as shown in Fig. 3.13 in Prolog:

```
pre_rule(for,2,b,[4,1]) :-
   stone(2,[5,1],w),
   stone(1,[6,2],b),
   stone(-3,[5,2],w),
   stone(-2,[4,2],b),
   stone(-1,[5,3],b),
   edge([6,0]),
   edge([5,0]),
   not stone(_6430,[6,1],w),
   not stone(_6458,[7,1],w),
   not stone(_6642,[4,1],w).
```

Figure 3.13: Pre-rule in Prolog.

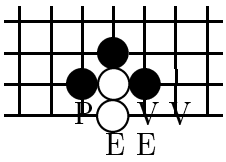

Figure 3.14: Rule f2. It is acquired by generalizing pre-rule in Step 11. **E** stands for an edge.

In Step 11, the pre-rule (Fig. 3.12) is generalized as shown in Fig. 3.14. E in the figure represents that the place should be an edge and decides the relative position of the stones to edges.

In Step 12, the generalized rule is registered to Knowledge Base as the second forcing rule (Rule f2).

All the steps so far are described in Fig. 3.15.

After acquiring the rule, Loop 1 continues and Step 1 is executed again. Now since the system can interpret ❸ as application of Rule f2 for attack by the winner, i_3 =`[3,f2,a,w]`. From Step 2 to 7 are in the same process mentioned above.

② is interpreted as application of Rule f1 for defense by a loser, thus i_2 = `[2,f1,d,l]`.

Since ❶ cannot be interpreted neither by Rule f1 nor by f2, another rule, Rule f3, which is shown in Fig. 3.17 is acquired. This is a well known position for all Go players including beginners.

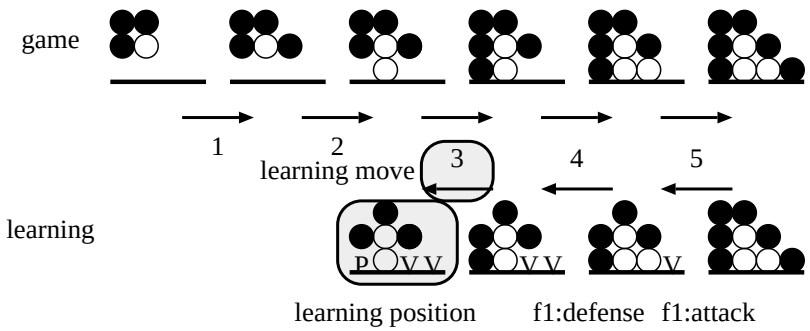

Figure 3.15: A process of extracting relevant parts.

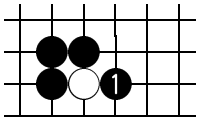

Figure 3.16: Board position on Move 1. $\delta(BC_0, s_1) = BC_1$.

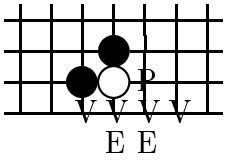

Figure 3.17: Rule f3.

The whole process of the rule acquisition is shown in Fig. 3.18.

3.5.4 Validity of Rule Acquisition Algorithm

In this section, the validity of Rule Acquisition Algorithm is discussed. In the first place, the validity of each step is discussed; the validity of Step 10 (the move to be learned) and Step 11 (generalizing rules) is discussed. Then, the validity of the whole process and the acquired rules is discussed.

Validity of the move to be learned

One of the most difficult things in acquiring rules in the domain of adversary games is to find good moves to be learned. In adversary games, a loser must have made some mistakes or bad moves. In other words, the loser failed to notice the good moves of the winner. Therefore, good moves which the system has not known must be specified in order to learn them.

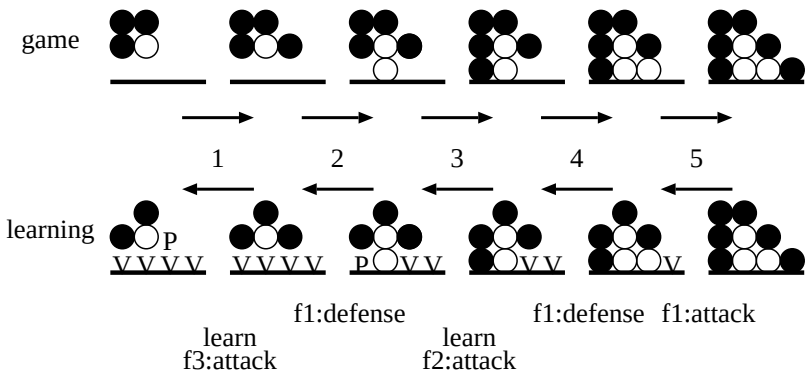

Figure 3.18: A process of rule acquisition: the above is a process in playing, and the below is that in learning.

In Rule Acquisition Algorithm (Section 3.5.2), a rule is acquired when Loop 2 is over and the move just before is learned. That is, the end of the interpretation sequence must be a defense move by a loser.

In the interpretation sequence, the first rule must be the one applied for attack by a winner and the rest of the rules must be applied alternately – for defense by a loser and for attack by a winner. The last rule in the sequence must be the one applied for defense by a loser.

The move made just before the interpretation sequence is actually the one which is the effect of application of a forcing rule for attack; the loser had to defend in all the turns thereafter. Thus the move and the position where it was made should be learned.

More precise explanation of the validity of the moves to be learned are as follows. Since this system learns rules when it loses, note that a loser is always this system in the following explanation.

In the following explanation, we suppose that all the forcing rules are proper, i.e. when the forcing rules are applied, the one who applies them will win.

Lemma 3.1 *When a loser applies forcing rules, they are always applied for defense.*

If a person applies a forcing rule for attack, he/she will win as the definition of forcing

rules in Section 3.3.2. Therefore a loser must not apply forcing rules for attack.

However, there are two possibilities that this lemma fails.

One possibility is that a person who applies forcing rules for attack intentionally or unintentionally does not continue to apply them for attack when possible, and loses. This case, however, does not occur because we only consider the case that a loser is the system as mentioned above. When the system applies a forcing rule for attack, it will continue to apply forcing rules for attack and it will win, as the definition of forcing rules.

The other possibility is that the rules applied are not appropriate. If the system applies forcing rules for attack and cannot win, these rules are not correct. These rules are refined as mentioned in Section 3.6.

Validity of generalizing rules (Step 11)

Generalization of colors is valid, because in the game of Go what Black can do goes for White.

In the game of Go, in capturing stones, only the relating position to the edge of the board is essential. It only depends on the related position to the edge whether stones are captured or not. Therefore the generalization of coordinates is valid.

In this step, only parallel movements are considered. In Go, it is always true that rotations and symmetrical movements about X, Y axes and lines $X = Y, X = -Y$, do not change the meaning of situation. These are not considered in this system.

3.5.5 Other results of rule acquisition

In this section, other examples are taken and the results are shown.

Geta

Geta is a famous tesuji [3], a net trap, which is learned by novices (Fig. 3.19). Although this is not an immediate threat and more difficult than the previous example, this algorithm acquires a proper rule as shown in Fig. 3.20.

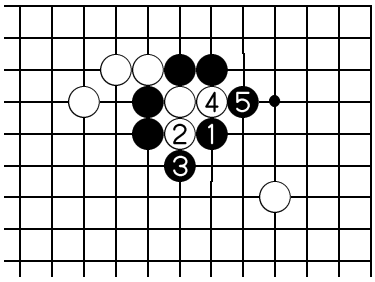

Figure 3.19: A problem and its answer of geta.

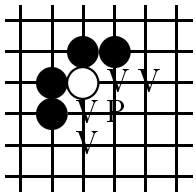

Figure 3.20: An acquired rule of Geta.

Ashida

This is a variant of Geta and more difficult than Geta (Fig. 3.21). This algorithm also acquires a proper rule of Ashida (Fig. 3.22).

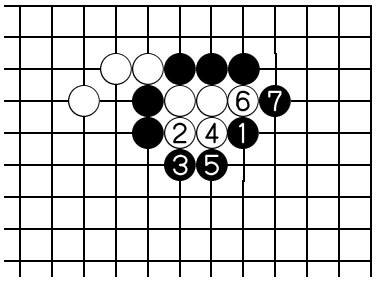

Figure 3.21: A problem and its answer of ashida.

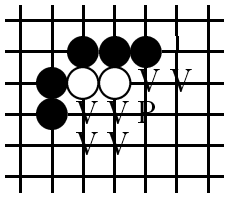

Figure 3.22: An acquired rule of Ashida.

[3]The best play in a local position; a skillful tactical move.

3.6 Rule Refinement

In adversary games, it is generally difficult to acquire proper rules from only one example. While a search tree in non-adversary games has only OR nodes, that in adversary games has AND nodes besides OR nodes(See Fig. 3.23. OR nodes has an arc.). In order to see whether a sequence of moves will reach the goal, only the direct path from start to the goal is concerned in non-adversary games. On the other hand, in adversary games, at AND nodes *all* the opponent's moves should be concerned including those which are not made in an actual game. In Fig. 3.23, the bold paths are those which should be concerned. Moreover, in adversary games, the opponent's moves should be also concerned which become possible as the consequence that some parts are removed from the position when rules are acquired. For the reason given above, to acquire precise rules from an example is difficult in adversary games.

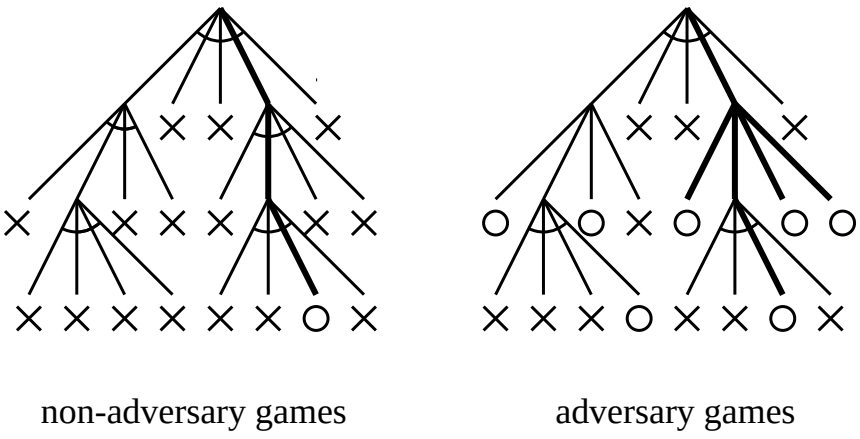

Figure 3.23: A difference of a search tree between of non-adversary games and of adversary games. ∘ denotes success, × failure.

Take Fig. 3.24 for example. Rule Acquisition Algorithm acquires the rule as the right of the figure from an example as the left of the figure. The right lacks the stone denoted **key stone**, which is necessary for Black to capture White, that is, the algorithm over-generalizes the left of the figure. One reason for over-generalization in this example is

that the moves actually made do not have a direct concern with the 'key stone' denoted in the figure; the importance of the **key stone** cannot be noticed unless the stone was removed. It is very difficult to notice the importance of the stone from only one example, which is true for humans(See Section 4.2.).

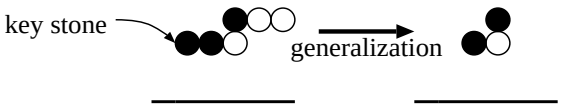

Figure 3.24: An example of over-generalization.

As can be seen, Rule Acquisition Algorithm may acquire over-generalized rules from an example. Instead of trying to acquire proper rules from a single example, this system refines acquired rules. It is explained by Fig. 3.25 and 3.26. In the figures, the largest oval, which contains some circles and/or an oval is the region where the rules hold.

Rules acquired by the algorithm mentioned above should be the sufficient conditions of the goals; if Board Configuration suffices the rule, the applier should win. Fig. 3.25 shows the normal rule acquisition. Rule f1, f2 and f3 are within the region, thus the acquisition succeeds.

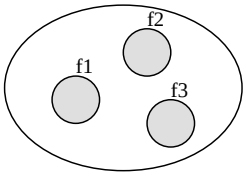

Figure 3.25: Normal Rule Acquisition.

In Fig. 3.26, Rule f2 is beyond the largest oval, i.e. in some situations which suffice Rule f2, application of Rule f2 for attack will not lead to win. Therefore the region of Rule f2 is made smaller by adding a restriction so that Rule f2 is within the largest oval.

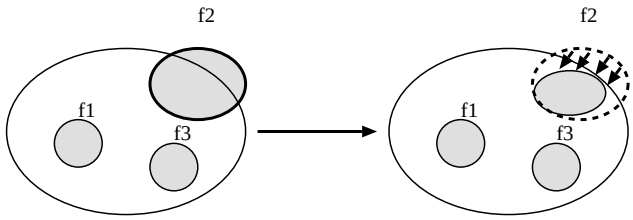

Figure 3.26: Rule Refinement by adding a restriction.

3.6.1 Rule Refinement Algorithm

The entrance of Rule Refinement Algorithm is Step 3 of Rule Acquisition Algorithm. The algorithm is shown below:

Algorithm 3.3 Rule Refinement

input from Rule Acquisition = $m, i_{m+1}, RP_m, RP_{m+1}, t_m, t_{m+1}, j$(=the last number of forcing rules)

1 **if** $i_{m+1} \neq$ `[m+1,f*,a,w]` **then exit Algorithm**

2 $L \leftarrow \{v_{x,y} | v_{x,y} \in RP_{m+1}\} \cup \{s_k | s_k \in RP_{m+1} \wedge s_k \notin RP_m\}$ $(L = \{l_1, \ldots, l_I\})$

3 $i \leftarrow 1$

repeat

4 **if** $l_i = v_{x,y}$ **then** $r_i \leftarrow s(= \langle [x, y], t_{m+1} \rangle)$

5 **if** $l_i = s(= \langle [x, y], t_m \rangle)$ **then** $r_i \leftarrow v_{x,y}$

6 $i \leftarrow i + 1$

until $i = I + 1$

7 **if** s_{m+1} has 2 liberties $([x_1, y_1], [x_2, y_2])$

then $r_{I+1} \leftarrow s = \langle [x_1, y_1], t_{m+1} \rangle, r_{I+2} \leftarrow s = \langle [x_2, y_2], t_{m+1} \rangle, I \leftarrow I + 2$

Loop

8 **if** $i_m \neq$`[m,fp,a,l]` **then exit Loop**

9 KB $\leftarrow$ KB $- rule_p^{for}$

10 $i \leftarrow 1$

repeat

11 pre-rule $\leftarrow \langle RP_m + r_i \rightarrow s_m \rangle$

12 $rule^{for}_{j+1} \leftarrow$ Generalize(pre-rule)

13 $j \leftarrow j + 1$

until $i = I + 1$

14 **if** $i_{m-1} \neq$ [$m-1$,f*,d,w] **then exit Loop**

15 $m \leftarrow m - 2$

end Loop

3.6.2 Explanation of Rule Refinement Algorithm

In general, Interpretation Sequence is denoted as below:

[[M,f1,a,w], [M-1,f1,d,l], [M-2,f*,a,w], [M-3,f*,d,l], ... , [2,f*,a,w], [1,f*,d,l]] (S1)

In some cases, Interpretation Sequence may be as follows:

[[M,f1,a,w], [M-1,f1,d,l], ..., [$p+2$,ft,a,w] [4], [$p+1$,f*,a,l] , [p,f*,d,w], ..., [1,fs,a,l]] (S2)

In Sequence S1, while a winner always applies forcing rules for attack, a loser always applies them for defense. In Sequence S2, a loser sometimes applies forcing rules for *attack* (from Move 1 to Move $p+1$). The rules applied by the loser for attack need to be refined, because they do not meet the definition of forcing rules; application of forcing rules for attack did not lead the loser to win. A sequence of the moves which yields Interpretation Sequence as S2 is called a *negative example* for the forcing rules.

Sequence S2 denotes the situation that though the system applies forcing rules for attack the system loses in reverse. Although there are many possibilities that need rule refinement, only the situation as Sequence S2 is concerned in this study (see Section 3.6.4).

The system refines rules by specifying the reasons why the system does not win in spite of applying forcing rules for attack; the reasons are called *failure reasons* in this

[4]Rule ft is acquired by Rule Acquisition Algorithm after a game, so the system can not notice during a game that Rule fs in the last element of Interpretation Sequence will cause the system to lose.

study. In some cases, a failure reason is that a winner applies a forcing rule for attack to system's stones which are necessary for the applications of rules for attack by the system.

After finding the failure reason, the rules are refined by adding a restriction which prevents the opponent from applying the failure reason.

In Sequence S2, a loser applies a forcing rule for attack on Move $p+1$ and a winner also applies for attack on Move $p+2$. Thus, the failure reason here is that the attacking move, Move $p+2$ is available for the opponent (winner).

The entrance of Rule Refinement Algorithm is Step 3 of Rule Acquisition Algorithm:

if i_m=`[m,f*,a,l]` **then Refinement**

As i_m is the interpretation that a loser applies forcing rules for attack, rules applied before Move m may be refined. m is $p+1$ in Sequence S2 and i_{m+1} in Step 1 is $p+2$ in Sequence S2. Since Move i_{m+1} is the failure reason, i_{m+1} should be `[m+1,f*,a,w]`.

Those rules will be applicable if Move $p+2$ in S2 is not available for the opponent. Since Move $p+2$ in S2 is Move $m+1$ in the algorithm, RP_{m+1} is the relevant parts to winner's attack and RP_m is the relevant parts to loser's attack. Therefore relevant parts which is in RP_{m+1} but not in RP_m is substituted into L in Step 2. In this process, all the vacant places are substituted. The reason is that vs in RP_{m+1} and in RP_m have different meaning since v means that the place where the opponent should not be placed.

L contains the parts unfavorable for the player who applies rules for attack which are now the targets for refinement. Thus l_i should be denied. Step 4 puts a stone on a vacant place whose color is the same as a loser so that a loser will win by applying the rule, which means as follows. If the place is vacant, the winner can attack. Thus the place should be protected by loser's stone. Step 5 removes winner's stone and changes the place into the vacant place, which means as follows. If there is the winner's stone, a loser cannot win. Thus the place should be vacant.

In Step 7, stones are added so that the winner cannot make the failure reason (s_{m+1}). If the stone has only one liberty after putting it, the loser can capture it and does not

lose.

Step 9 retracts a rule which is applied by the loser for attack. Step 11 makes a pre-rule which adds a restriction, r_i and it is generalized by Step 12, which is the same procedure in Step 11 of Rule Acquisition Algorithm.

Step 14 checks whether Move $m-1$ is still in Interpretation Sequence. Step 15 changes the focus from Move m to Move $m-2$ and Step 8 checks whether the loser applies a rule for attack; if this is true, the rule is also refined.

3.6.3 Examples of Rule Refinement

Rule Refinement Algorithm refines rules in two ways: adding a stone and adding a vacant place. In this section, these two examples of refinement are explained.

An example of adding a stone.

Take Figure 3.27 as an example of refinement by adding a stone.

The board positions on the left of the figure (from P0 to P9, a positive example) yield Interpretation Sequence as S1, thus a new rule is acquired whose IF-part, or condition-part is position N0 in the figure. In the middle of the figure, the system (Black) applies the new rule to Position N0 and fails to capture White stones by being captured by the opponent. The interpretation sequence for the example is like Sequence S2. Since the system (loser) applies rules for attack from position N0 to N5, the rules applied to the positions should be refined.

Position N5 is the result of attack by a loser (Black), and Position N6 is that of attack by a winner (White) to Black stones which are necessary for capturing White three stones surrounded by Black stones. Therefore the failure reason here is that White can attack on Move 6.

The algorithm adds a restriction to the rules which the loser applied for attack so that White cannot apply the forcing rule on Move 6. The differences between Relevant Parts

of Move 5 (RP_5) and those of Move 6 (RP_6) are some vacant places. Thus the results of the refinement by adding stones on the vacant places are shown on the right side of the figure.

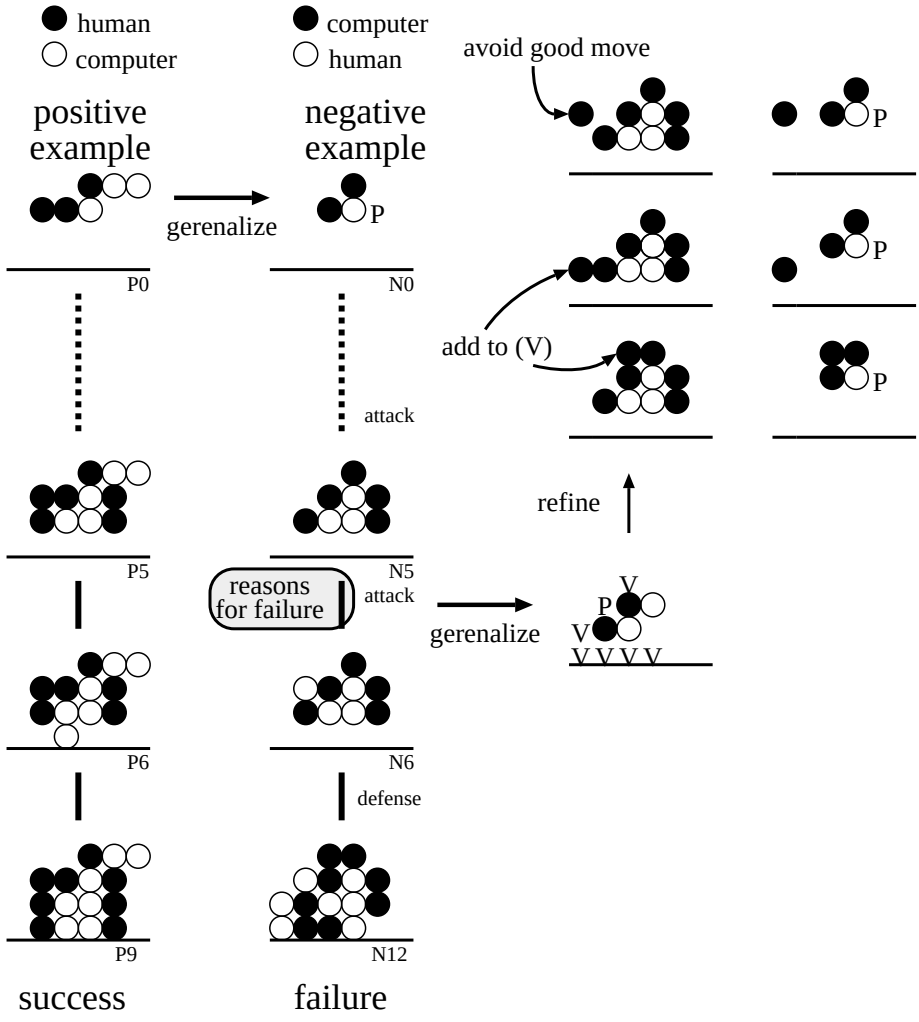

Figure 3.27: An example of refinement of an acquired rule.

An example of adding a vacant place

In the previous section, an example of the rule refinement by adding stones is explained. In this section, an example of the refinement by adding a vacant place is introduced.

Fig. 3.28 is an example of a problem and Fig. 3.29 is a rule acquired from the example.

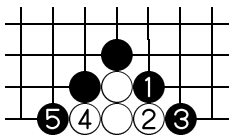

Figure 3.28: A problem and its solution.

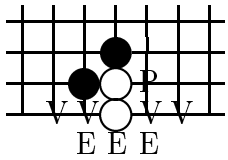

Figure 3.29: A rule acquired from Fig. 3.28.

Now in Fig. 3.30, a White stone marked **a** is added to Fig. 3.28. Black fails to capture White stones and is captured in reverse. In this example, $m = 1$.

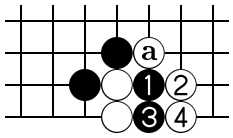

Figure 3.30: A negative example for the rule in Fig.3.29. A White stone marked **a** is added.

Relevant Parts on Move 2 (RP_2) are shown in Fig. 3.31 and those on Move 1 (RP_1) in Fig. 3.32. White stone which is in Fig 3.31 and which is not in Fig 3.32 is a stone on [5,3], thus a vacant mark is added to the position in the left of Fig. 3.33. Black stone is added in Fig 3.33 on three vacant places in 3.31.

3.6.4 Discussion on Rule Refinement Algorithm

Since forcing rules are the sufficient conditions for capturing opponent's stones, all the rules should be refined that are applied for attack but do not make the applier win. In

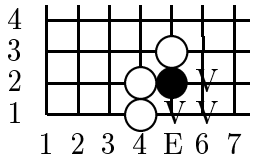

Figure 3.31: Relevant Parts on Move 2 (RP_2).

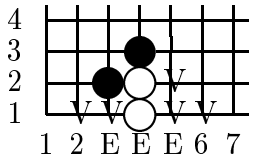

Figure 3.32: Relevant Parts on Move 1 (RP_1).

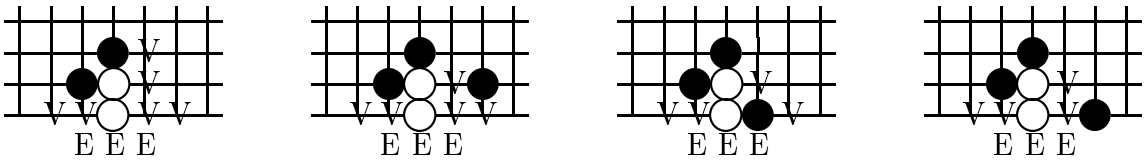

Figure 3.33: Rules acquired from Fig. 3.30. The most left is by adding a vacant place and the right three are by adding a stone.

this study, rules are refined when the system tries to capture opponent's stones as forcing rules suggest but the system's stones are captured in reverse.

In general, however, there are many other possibilities that need rule refinement. Take a move which is both application of a forcing rule for defense and *ate*, a move which makes opponent's stones in atari, for example. This move is a defense but the opponent must defend in the next move because the move is also *ate* and the opponent's stones are in atari. In this case, the applier of forcing rules for attack changes, which means some rules should be refined. However, those rules cannot be refined because the move is not an application for attack.

This will be able to be learned when a concept, *threat* is introduced. Details in threat is mentioned in Section 5.4.

Chapter 4

Cognitive Experiments

In this chapter, the behaviors of this model and those of human beginners are compared. Beginners are classified into three categories as mentioned in Section 1.2.1: the unexperienced, pre-novices, novices.

4.1 Comparison of Rule Acquisition Algorithm with humans

In this section the experiments to compare Rule Acquisition Algorithm mentioned in Section 3.5 (this model) with humans are explained.

4.1.1 Observation of humans – a preliminary experiment

In the first place, two novices are observed when they solve and learn life and death problems and stone-capturing problems. During this observation, they sometimes try to find their mistakes by tracing back from the solution position to the initial position.

According to this observation, the model is proposed as mentioned in Section 3.

4.1.2 A cognitive experiment

The object of this experiment is to investigate the learning methods of the unexperienced and pre-novices. Two learning phases are assumed: a *natural learning phase* and a *instructed learning phase*. The natural learning phase is that to investigate the same learning method as they usually do. The instructed learning phase is that to investigate whether they are able to learn rules deductively as this model when they are indicated implicitly to learn rules by the same method as this model.

Procedure of this experiment

Subjects are three pre-novices and two unexperienced persons. This experimental procedure is as follows:

1. Subjects fill in a face sheet. Total hours when they play Go are asked.

2. Rules of Go are explained. The object of this step is to let subjects know that stones are captured when they have no liberties.

3. For the unexperienced, thirty easy problems are given so that they becomes accustomed to Go rules.

4. Each problem which is Black to move is shown and should be replied within 5 seconds from three choices (Subject Choice, SC):

   - Black can capture White stones. (capture,$\circ$)
   - Black cannot capture White stones. (not capture, $\times$)
   - I don't know. (not know, ?)

   Subjects are instructed that unless they are sure they should replay "not know". There is no feedback; the reply is correct or not is not told to the subjects. Two sets of problems (one set consists of 18 problems), called **test problems**, are shown in

order to check the subjects' consistency. 18 problems look similar with each other. In this step, the knowledge before learning is checked.

5. A problem, called a **learning problem** is given. Subjects try to solve it putting stones. When they reach the solution, they are told that they are correct. After they solve it, they are asked to think why White stones are captured by Black. The correct reason is not told. This phase is called a **natural learning phase**.

6. The same step as Step 4. This is the result of the natural learning phase.

7. The learning problem is given. Subjects are instructed to show the unnecessary Black stones for capturing White stones. They are also instructed to show the place where White stones should not be placed in order to capture White stones. Their solutions are not told. This phase is called a **instructed learning phase**.

8. The same step as Step 4. This is the result of the instructed learning phase.

In short, Step 4 tests the pre-knowledge by 18×2 test problems. Step 5 is the natural learning phase and subjects learn a learning problem. Step 6 tests the result of the natural learning phase by the test problems. Step 7 is the instructed learning phase and subjects learn the learning problem. Step 8 tests the results of the instructed learning phase by the test problems.

Details of problems used in this experiment

Since Rule Acquisition Algorithm (this model) is supposed to acquire the *sufficient* conditions to capture opponent's stones, the test problems used in Step 4, 6 and 8 can be classified into three categories:

$$(\text{Correct Solution } (CS), \text{ Suggestion of Model } (SM)) = (\circ, \circ), (\circ, \times), (\times, \times)$$

The number of problems are 5, 7 and 6, respectively.

Figure 4.1 is the learning problem, used in Step 5 and 7, and a rule acquired by this model is as Figure 4.2, in which the place with **V** should not be placed by White stones and that with no mark is either Black, White or a vacant place.

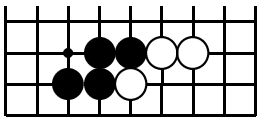

Figure 4.1: The learning problem.

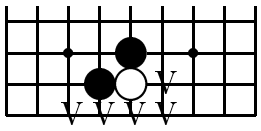

Figure 4.2: The model suggestion of the learning problem.

Examples of $(CS, SM) = (\circ, \circ), (\circ, \times), (\times, \times)$ are Figure 4.3, 4.4 and 4.5, respectively.

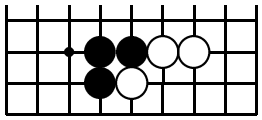

Figure 4.3: An example of $(CS, SM) = (\circ, \circ)$.

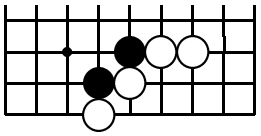

Figure 4.4: An example of $(CS, SM) = (\circ, \times)$.

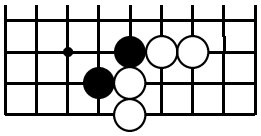

Figure 4.5: An example of $(CS, SM) = (\times, \times)$.

Another classification is considered. In most Case-Based Reasoning(CBR) [16, 27] systems, simple similarity indexes are used. Thus similarity (Sim) is the sum of the number of different places from the learning problem (Figure 4.1). For example, Sim's of Figure 4.3, 4.4 and 4.5 are 1, 3 and 3, respectively.

Results of this experiment

n is a function which returns the number.

Consistency is the rate of consistency between two sets. It is calculated as follows:

$$\text{Consistency} \equiv \frac{n(c_1 = c_2)}{n(toal)},$$

where c_1 is the first choice of a problem and c_2 is the second one.

The correct rate (CR) is calculated as follows:

$$CR \equiv \frac{n(CS = \circ, SC = \circ) + n(CS = \times, SC = \times)}{n(total)}$$

Similarity Coincident Rate (SCR) is calculated as follows:

$$SCR \equiv \frac{n(0 \leq Sim \leq 2, SC = \circ) + n(3 \leq Sim \leq 5, SC = \times)}{n(total)}$$

Another Similarity Coincident Rate (SCR') is calculated as follows:

$$SCR' \equiv \frac{n(0 \leq Sim \leq 2, SC = \circ) + n(3 \leq Sim \leq 5, SC = \times)}{n(SC = \circ) + n(SC = \times)}$$

The model coincident rate (MCR) is calculated as follows:

$$MCR \equiv \frac{n(SM = \circ, SC = \circ) + n(SM = \times, SC =?) + n(SM = \times, SC = \times)}{n(total)}$$

The tables below are the results of five subjects, which are named from A to E in order of consistency. The tables from A to C are those of three pre-novices and D and E are those of two unexperienced person.

CS	SM	SC of Step 4 ∘	?	×	SC of Step 6 ∘	?	×	(SC of Step 8) ∘	?	×
∘	∘	2	8	0	8	2	0	5	0	0
∘	× or ?	0	11	3	0	12	2	7	0	0
×	× or ?	0	11	1	0	2	10	3	0	3
$0 \leq Sim \leq 2$		0	19	1	8	6	6	8	0	2
$3 \leq Sim \leq 5$		2	11	3	0	10	6	7	0	1
Consistency (%)		88.9			88.9			—		
CR (%)		8.3			50.0			83.3		
SCR (%)		8.3			38.9			50.0		
SCR' (%)		50.0			70.0			50.0		
MCR (%)		77.8			94.4			(44.4)		

Table 4.1: Results of Subject A.

CS	SM	SC of Step 4 ∘	?	×	SC of Step 6 ∘	?	×	SC of Step 8 ∘	?	×
∘	∘	9	1	0	8	2	0	8	2	0
∘	× or ?	6	7	1	5	9	0	2	12	0
×	× or ?	6	1	5	6	5	1	6	0	6
$0 \leq Sim \leq 2$		14	2	4	14	5	1	12	4	4
$3 \leq Sim \leq 5$		7	7	2	5	11	0	4	10	2
Consistency (%)		66.7			88.9			100		
CR (%)		55.6			38.9			44.4		
SCR (%)		44.4			38.9			44.4		
SCR' (%)		59.3			70.0			63.4		
MCR (%)		63.9			63.9			72.2		

Table 4.2: Results of Subject B.

CS	SM	SC of Step 4 ∘	?	×	SC of Step 6 ∘	?	×	SC of Step 8 ∘	?	×
∘	∘	2	0	8	6	2	2	8	0	2
∘	× or ?	8	0	6	12	0	2	11	1	2
×	× or ?	2	1	9	4	0	8	7	0	5
$0 \leq Sim \leq 2$		2	1	17	11	2	7	14	1	5
$3 \leq Sim \leq 5$		10	0	6	11	0	5	12	0	4
Consistency (%)		83.3			88.9			77.8		
CR (%)		52.8			72.2			66.7		
SCR (%)		22.2			44.4			50.0		
SCR' (%)		22.9			47.1			51.4		
MCR (%)		50.0			44.4			44.4		

Table 4.3: Results of Subject C.

CS	SM	SC of Step 4 ∘	?	×	SC of Step 6 ∘	?	×	SC of Step 8 ∘	?	×
∘	∘	0	8	2	9	0	1	10	0	0
∘	× or ?	1	7	6	5	3	6	7	2	5
×	× or ?	1	3	8	2	3	7	10	1	1
$0 \leq Sim \leq 2$		1	8	11	10	5	5	17	1	2
$3 \leq Sim \leq 5$		1	10	5	6	1	9	10	2	4
Consistency (%)		61.1			44.4			61.1		
CR (%)		25.0			58.3			50.0		
SCR (%)		16.7			52.8			58.3		
SCR' (%)		33.3			63.3			63.6		
MCR (%)		66.7			77.8			52.8		

Table 4.4: Results of Subject D.

CS	SM	SC of Step 4 ∘	?	×	SC of Step 6 ∘	?	×	SC of Step 8 ∘	?	×
∘	∘	1	2	7	6	0	4	5	0	5
∘	× or ?	1	3	10	6	0	8	6	0	8
×	× or ?	4	0	8	8	0	4	5	0	7
$0 \leq Sim \leq 2$		5	1	14	12	0	8	10	0	10
$3 \leq Sim \leq 5$		1	4	11	8	0	8	6	0	10
Consistency (%)		50.0			55.6			44.4		
CR (%)		27.8			44.4			50.0		
SCR (%)		44.4			55.6			55.6		
SCR' (%)		51.6			55.6			55.6		
MCR (%)		61.1			50.0			55.6		

Table 4.5: Results of Subject E.

Considerations of the results of the experiment

The consistencies of pre-novices are relatively high. Subject A seems to learn almost the same rule as this model as early as in the natural learning phase, because MCR rises from 77.8 % to 94.4 %. Subject B seems to learn the similar rule as this model in the instructed phase, because MCR rises from 63.9 % to 72.2 %. In the results of both subjects, SCR

and SCR' are not monotonous, thus they seem not to acquire knowledge by the simple similarity. The result of Subject A in Step 8 is different from those of other subjects, because Subject A is differently instructed from other subjects; since Subject A acquires the rule same as this mode, he is instructed to solve other problems which are given by himself.

Subject C seems to follow neither correct solution nor the model.

The consistencies of the unexperienced person, Subject D and E are relatively low. In Step 6 and 8, the consistency of Subject E is around 50 % even though his choices are $\circ$ and $\times$, which means that his choices are almost at random. As the consequence, it seems hard for the unexperienced to learn.

In the natural learning, simple similarity-based learning as CBR may exist. In Subject D, SCR rises from 16.7 % to 52.8 % and SCR' does from 33.3 % to 63.3 %. In Subject E, SCR rises from 44.4 % to 55.6 % and SCR' does from 51.6 % to 55.5 %. In the instructed learning, SCR and SCR' of both subjects does not change. Thus, it may be that the natural learning includes similarity-based learning and the instructed learning does not. More experiments are needed to clarify these points.

Therefore, pre-novices may be able to acquire rules as this model, while it is hard for the unexperienced to learn in the way as this experiment.

4.2 Comparison of Rule Refinement Algorithm with humans

Rule Refinement Algorithm mentioned in Section 3.6 is compared with human novices.

4.2.1 Experiment 1

Subjects are two novices. They are given the same position as the positive example in Figure 3.27 (Figure 4.6) and asked to remain the least number of stones which are needed

in order to capture White stones. They were not sure whether they could capture White stones without the key stone on [4,3]. This result suggests that human novices may make the same mistakes as Rule Acquisition Algorithm does.

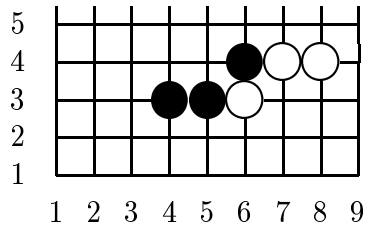

Figure 4.6: The given problem in Experiment 1.

4.2.2 Experiment 2

Moreover, the same novices are given the negative example in Figure 3.27 (Figure 4.7) and are shown the process of failing to capture White stones (Figure 4.8). Then they are asked to add Black stones in order to capture White stones.

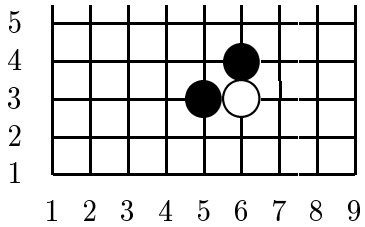

Figure 4.7: The given problem in Experiment 2.

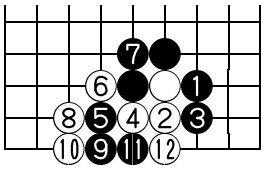

Figure 4.8: The process given in Experiment 2.

They realized that the reason for Black's failure is White's 6th move, ⑥, which is both *ate* and a *threat* to capture the Black stone, ❺. Thus they put each stone to avoid these threat. The results of the subjects are shown in Fig. 4.9, and those of Rule Refinement Algorithm are shown in Fig. 4.10.

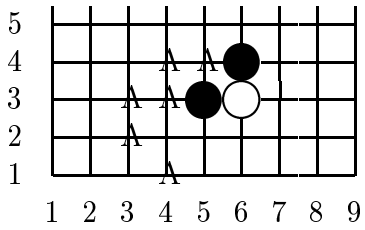

Figure 4.9: The results of subjects of Experiment 2.

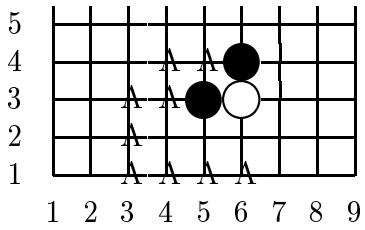

Figure 4.10: The results of Rule Acquisition Algorithm of the same problem as Experiment 2.

The results of humans and those of Rule Acquisition Algorithm are much the same except the first low.

Chapter 5

Discussion

5.1 Comparison with other learning algorithms in adversary games

There are two major approaches to building learning systems in the domain of adversary games: a numerical (parameter) learning approach and a knowledge acquisition one. Most learning systems in adversary games take the former approach, which usually changes the evaluation functions, which are used in brute force search. The latter approach is adopted in knowledge-intensive systems. The proposed system takes the knowledge acquisition approach in the knowledge-intensive system.

5.1.1 Numerical learning in adversary games

A classical and the most famous program which learns numerically is Samuel's checkers program [31, 32]. This program has a static evaluation function, which evaluates the whole board. Brute force search chooses the move whose value of the evaluation function is maximized. This program changes the way to calculate the function. The evaluation

function consists of some parameters, each of which evaluates a signature [1] of the board. This program changes the parameters referring to the moves in the given games.

This method has two problems when applied to Go: the brute force search and the evaluation function.

First, this method assumes that system executes brute force search to select the best move during a game. Perfect evaluation functions are impossible to be generally defined in adversary games, which are functions whose value of a move is the same as that which results from two-ply minimax search. Thus, moves are chosen by calculating deeper, which means that all the moves are evaluated in a certain moves, say m depth. Since the branching factor of Go is usually over 200, even if the evaluation function needs the same time of calculation, m is at most 3, which is far from satisfactory.

Secondly, this method assumes that the static evaluation function of the whole board can be defined by the combinations of signatures of the board. In chess or checkers, loss or gain of pieces is the main signature and many other signatures can be numerically evaluated without much difficulties. In Go, on the other hand, only a small number of signatures can be defined. Moreover, calculation of the signatures in Go which can be defined is much more expensive than that in chess or checkers. One of the signatures in Go is the points of the field which a player is sure to gain. Calculation of this signature is almost 1000 times as expensive as that of loss or gain of pieces in chess [36].

Because of both problems, the numerical learning is not appropriate for learning Go.

Other numerical learning programs are listed below. Smith [39] builds a learning program which plays four-player partnership dominoes, which adopts the same method of Samuel's program. Tesauro [40] builds an intermediate-to-advanced level backgammon program using neural networks which are trained by back-propagation learning. Lee and Mahajan [17] builds a world class Othello program, BILL applies Bayesian learning to combine features in BILL's evaluation function. All of the programs listed above are

[1] A feature of the board such as mobility of pieces, or the gain of materials

difficult to be applied to Go because of the same reasons as those for Samuel's program.

5.1.2 Other knowledge acquisition systems in adversary games

Since systems which acquire some knowledge in Go have not been built, the differences between other knowledge acquisition systems in other adversary games and the proposed system are discussed in this section.

Small number of knowledge acquisition systems have been built in this domain, and few have been built which seem to be applied to other games. Among these programs, Minton's go-moku system will be discussed here, which is the typical system of knowledge acquisition in adversary games.

S. Minton [23,24] presents a go-moku[2] program using Constraint-based Generalization. Although the algorithm for rule acquisition mentioned in Section 3.5 is similar to Minton's one in purpose, they are different in requirements for learning and in refinement.

Minton's program has two kinds of rules: State-Update rules and recognition rules. State-Update rules are provided by the programmer for adding and deleting features from Game-States after each turn, and they are indispensable to learning rules [23]. Recognition rules are similar to the forcing rules of the proposed system and described by State-Update rules.

Although State-Update rules are easily defined in go-moku domain, they are difficult to be defined in Go domain; while in go-moku only one goal state is specified and enough terms are provided to describe states of positions such as 'three-in-a-row', in Go there are immense positions where stones are captured[3] and sufficient terms are not provided to describe states of the positions. Figure 5.1 shows three examples of the positions.

State-Update rules are used to acquire more generalized rules. Since they are hard to

[2]A variant of tic-tac-toe. The goal of the game is to get five stones in a row either horizontally, vertically or diagonally.

[3]This is the goal state in this system.

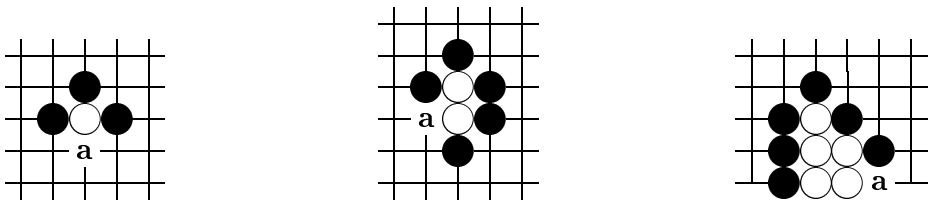

Figure 5.1: Examples of atari. All the White stones are in atari and **a** is the last liberties.

be defined in Go, Rule Acquisition Algorithm does not use State-Update rules but uses actual moves and positions appearing in process of a game besides forcing rules. Thus it is able to acquire Go rules. Although this system has less requirements, generality of acquired rules is less than that of Minton's. More generalized rules are considered in Shirayanagi's approach, which is discussed later.

Also, while Minton's system does not refine rules, the proposed system does. Minton does not refer to refinement, because in go-moku it is rarely required. Go is a much more situation-dependent game than go-moku; it is often the case with Go that a rule which is applicable in a situation is not always applicable in a very similar situation. Thus rule refinement is indispensable in Go and Minton's approach is not good enough to learn proper rules in Go.

5.2 Other knowledge representation in Go

Shirayanagi's approach to Go programs will be discussed here, which is not a knowledge acquisition system but a knowledge-intensive Go program which is different from this system in knowledge representation.

Shirayanagi's approach to Go programs [38] is to represent knowledge not *arithmetically* but *symbolically*, whereas almost all Go playing programs use evaluation functions to represent knowledge arithmetically. He proposes that knowledge is represented in tree

expressions whose leaves are Go terms such as *ate* [4]. Rules are represented using Go terms such as "DO *hane* to *tsuke*", which means that when an opponent makes a move, *tsuke*, then do *hane*. The priority of rules is represented by tree structures. When an opponent puts a stone, his program names the move and decides a move according to rules tagged by the name.

This approach is similar to the study of Saito [30], where human players select the best move *verbally*[5] from candidate moves; they evaluate candidate moves using Go terms.

Although the author does not object to the hypothesis that human players use Go terms in evaluation of moves, the author makes the assumption that human players use both Go terms such as are used in Shirayanagi's approach and *shapes* [6] such as are acquired in the author's one. There are many shapes which have no name, on the basis of which players make a move.

Terms and shapes have different advantages and disadvantages.

An advantage of using Go terms in rules is that the rules are more general (i.e. rules written by terms have more cases to be applied), and a disadvantage is that there are more rules which are not explicitly described (i.e. there are more cases which cannot be described by terms).

On the other hand, an advantage of using shapes is that all patterns will be described by shapes, using stones and edges, for example, to acquire rules and a disadvantage is that the rules are less general (i.e. rules using shapes have less cases to be applied to than those written by terms).

The author is sure that it is the best to use both terms and shapes. They should be used in the right situations. By using both of them, they are considered to comply the weaknesses of each other.

Therefore, the ideal knowledge acquisition system in Go has rules described both by

[4] *Ate* is to make opponent's stones in atari.
[5] It means that human players select the best move using Go terms.
[6] They mean arrangements of stones.

terms and by shapes. Since this system acquires rules described by shapes, the system is the first step toward this ideal system.

5.3 Comparison with original EBG

This system partly uses Explanation-Based Generalization(EBG) [13, 25] in Rule Acquisition Algorithm. In this algorithm, EBG is used in order to extract parts relevant to an interpretation of a move from the whole board, which is made possible by the explanation step of EBG (Section 2.2). However, it has following three main differences from EBG: knowledge level learning, refinement and use of negative examples.

First of all, while EBG does symbol level learning, this system implements knowledge level learning. Dietterich [6] proposes that machine learning should be classified into two classes: symbol level learning and knowledge level learning. The former learning is that which does not acquire knowledge but changes the efficiency of calculation, and the latter one is that which acquires some knowledge. EBG is a typical example of symbol level learning. The reason that this system does knowledge level learning is that Go is an intractable theory problem, whose theory is complete but for which it is computationally prohibitive to construct explanations in terms of the theory [25]. Go is intractable because the definition of Go is complete but it is too expensive to calculate all moves to find the best move in the situation. Since Go is intractable, examples (moves of good players) play a major part in learning; it is too difficult to find good moves for a system itself but easy to learn when an opponent shows. Therefore, a new rule which cannot be found can be acquired when the system realizes that some opponent's moves are good.

Secondly, while EBG does not change the acquired rules, this system refines the acquired rules. Finally, while EBG does not use negative examples, this system does. In this system, some negative examples are used to refine the acquired rules. Since the original EBG does not refine rules, negative examples cannot be used.

The main reason for the differences is related to another use of EBG concept. The

concept of EBG is used globally; the process of making Interpretation Sequence by forcing rules is regarded as the similar process of making an explanation tree in explanation step of EBG. Interpretation Sequence explains globally why the winner wins.

Another reason for the differences between EBG and the method adopted in this system is that the target domain of this system is an adversary game domain. It is impossible in adversary games to make an explanation tree of a game because the situations completely change after a move. After a player makes a move, what was good for the player changes into what is bad for the other player. Therefore, a global EBG is necessary to bridge the gap after a move.

5.4 Future works

In this study, rules concerning the way to capture stones are acquired. However, more difficult way to capture stones such as *semeai*, or capturing race is not realized, which is the next target for future. Other Knowledge which novices know should be acquired for the system to play on a level with human novices.

Acquisition of a kind of category was observed in the protocol data of some novices. This should be also tackled in the future.

Learning more difficult way to capture stones

In *semeai* (or capturing race) problems, where both players try to capture opponent's stones, the priority of rules is necessary to be given. In this system, as mentioned in Section 3.4, all rules but Rule f1 have the same priority; if some rules can be applied in a situation, the rule acquired last is to be applied. In addition, attack is always prior to defense except for Rule f1. In some cases, while an attack takes 5 moves to capture opponent's stones, the opponent's attack takes 4. In this case, a player should defend opponent's attack, whereas this system attacks. A solution to this problem is to acquire

rules according to the degree of emergency, which tells how many moves are needed to capture stones. This can be implemented by counting the number of liberties of target stones.

A concept of *threat* will be useful for *semeai* problems, as mentioned in Section 3.6.4. Threat can be defined as a move, the result of which satisfies a IF-condition of a forcing rule. A typical threat is *ate*, a move which makes opponent's stones in atari. The concept threat can be used when the system interprets moves; moves can be interpreted by application of forcing rules and/or threats. As a consequence, a move which is both application of a forcing rule for defense and a threat will be interpreted properly and a rule concerning the move will be acquired.

Learning other knowledge for novices

Connectivity (whether a string is connected to the other) is very important concept in Go play. By defining the connectivity, this concept may be learned in the same algorithm. After learning this concept, the proposed program will be able to play 9×9 board game, whose battle is mostly local.

Acquisition of categories

Acquisition of a kind of category, such as shown in Figure 5.2, should be considered. One of the experiments suggests that some novices regard all the positions as belonging to the same category; if Black puts a stone on place **a** in each position, Black can take White stones by the same sequence of moves in any case. At present, however, this system regards such positions as different. Therefore the system must be extended to simulate such a recognition process – a process of categorization. The author is planning to make experiments to investigate such a process.

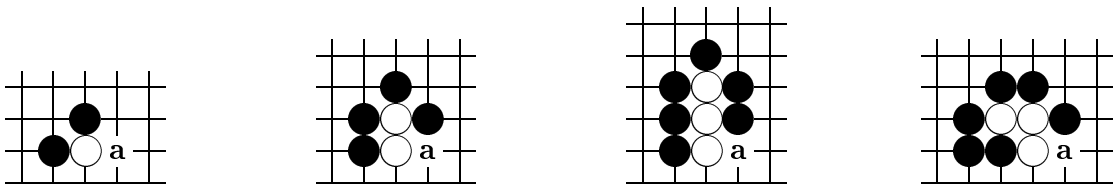

Figure 5.2: An example of positions which belong to the same category for some novices. **a** denotes the place where a stone should be put.

Implicit learning in Go

Although in this paper, deductive learning in Go is concerned, there may be implicit learning [3,35] in Go, too. Three criteria of implicit learning [35] are as follows: 1. the knowledge gained in implicit learning is not fully accessible to consciousness, 2. subjects learn information that is more complex than a single association or frequency count, and 3. implicit learning does not involve processes of conscious hypothesis testing.

Go seems to involve some learning which is true of the three criteria. First, Go experts have much knowledge which cannot explicitly described (Criterion 1). Most Go knowledge is, of course, more complex than a single association or frequency count (Criterion 2). In some cases, Go patterns are learned without calculating moves (Criterion 3).

Seger also says that implicit learning is robust over time. It is said that the decrease in the playing strength after long absence of playing is far less in Go than in chess or other games. Thus implicit learning may be found more frequently in Go learning than in chess.

Chapter 6

Conclusion

In order to build Go playing systems, knowledge intensive approach should replace brute force search, which has succeeded in chess. It is too difficult, however, to implement enormous patterns which human experts know and which should be implemented in terms of knowledge intensive approach, thus rule acquisition algorithms should be proposed in order to build stronger Go systems. However, there are four problems in the previous studies on building Go learning systems: existing pattern acquisition algorithms are less powerful, there are no rule refinement algorithms in adversary games, no learning systems have been compared with humans and the hierarchical structure of knowledge in most expert seems abnormal. In this study, these above problems are partly settled.

First, a new knowledge acquisition system is built to settle the first problem. This is the first system to acquire rules in Go. Minton [23] proposes a pattern acquisition algorithm in go-moku, which is not powerful enough to learn rules in Go, because State-Update rules need to be given, which are difficult to be defined in the domain of Go. The proposed algorithm does not need to be given State-Update rules, because it makes direct use of the positions and the moves appearing in process of a game.

This algorithm uses the concept of Explanation-Based Generalization (EBG). EBG explains the reasons why a given example is an example of the goal concept by making an explanation tree and extracts only the relevant features to the explanation. This

algorithm uses EBG in two ways: locally and globally. Local EBG explains a move and extracts the relevant parts of the board, which is based on the explanation step in EBG. Global EBG explains a sequence of moves to make a tree, which is called Interpretation Sequence. This is a new way to use EBG.

Secondly, a new refinement algorithm is proposed, which is the first attempt in adversary games. In case that a negative example against rules is given, this algorithm derives the reasons of the failure by analyzing Interpretation Sequence and adds a restriction to the rules so as to avoid the negative example.

Thirdly, these algorithms are compared with human beginners. This is the first attempt to compare algorithms with humans in adversary games. The first experiment indicates that two of the three human pre-novices acquire rules like this system, neither like CBR nor to learn correct solutions. The two unexperienced persons seem to behave almost randomly because their consistency of two sets of problems is about 50 %. The second experiment shows that human novices refine rules like this algorithm. These experiments indicate that human beginners may acquire and refine rules as this system.

This system partly uses EBG. This system, however, has three differences from EBG. First of all, while EBG does symbol level learning, this system realizes knowledge-level learning. Secondly, while EBG does not use negative examples, this system does. Finally, while EBG does not change the acquired rules, this system refines acquired rules. The reasons for the differences are that the target problem of this system is one of adversary games and that EBG is used globally and locally in this system.

Acknowledgment

I would like to express my gratitude to Prof. Saburo Nagano for his support as a supervisor throughout my years as graduate student.

I would like to thank Research Assistant Kazuhiro Ueda for his continuing guidance and his helpful suggestions with respect to the format of this thesis.

I wish to thank Go Expert, Masahiro Hachimori for his helpful advice on Go.

Thanks are due to Kiyoshi Izumi and Takashi Takabatake for their encouragement.

I want to thank all members of Nagano Laboratory for their useful discussion.

Bibliography

[1] H. J. Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14:205–220, 1980.

[2] H. J. Berliner and C. Ebeling. Hitech. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 6, pages 79–109. Springer-Verlag, 1990.

[3] D. C. Berry and Z. Dienes. *Implicit learning : theoretical empirical issues.* L. Erlbaum Associates, 1993.

[4] B. G. Buchanan and R. G. Smith. Foundations of expert systems. In A. Barr, P. R. Cohen, and E. A. Feigenbaum, editors, *The Handbook of artificial intelligence*, volume 4, chapter 18. Addison-Wesley, 1989.

[5] K. Chen, A. Kierulf, M. Müller, and J. Nievergelt. The design and evolution of Go explorer. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 17, pages 271–285. Springer-Verlag, 1990.

[6] T. G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1(3):287–316, 1986.

[7] E. A. Feigenbaum and J. Feldman, editors. *Computers and Thought.* McGraw-Hill, 1963.

[8] D. Fotland. Biography of the many faces of go. This is the article reachable by World Wide Web: `http://cgl.ucsf.edu/go/Programs/ManyFaces.html`.

[9] D. Fotland. Knowledge representation in the many faces of go. Manuscript available by Internet anonymous ftp from `basderver.ucsf.edu:Go/comp/mfg.Z`, 1993.

[10] F. h. Hsu, T. Anantharaman, M. Campbell, and A. Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4), October 1990.

[11] F. h. Hsu, T. S. Anantharaman, M. S. Campbell, and A. Nowatzyk. Deep Thought. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 5, pages 55–78. Springer-Verlag, 1990.

[12] R. M. Hyatt, A. E. Gower, and H. L. Nelson. Cray Blitz. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 7, pages 111–130. Springer-Verlag, 1990.

[13] S. T. Kedar-Cabelli and L. T. McCarty. Explanation-Based Generalization as resolution theorem proving. In *Fourth International Machine Learning Workshop*, pages 383–389, 1987.

[14] H. Kitano et al. Grand challenge AI applications. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1677–1683. Morgan Kaufmann, 1993.

[15] T. Kojima, K. Ueda, and S. Nagano. A case study on acquisition and refinement of deductive rules based on EBG in an adversary game: how to capture stones in Go. In *Game Programming Workshop in Japan '94*, pages 34–43. 1994.

[16] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[17] K.-F. Lee and S. Mahajan. The development of a world class othello program. *Artificial Intelligence*, 43:21–36, 1990.

[18] D. N. L. Levy, editor. *Computer Games*, volume I & II. Springer-Verlag, 1988.

[19] T. A. Marsland. Computer chess methods. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 159–171. Wiley, 1987.

[20] T. A. Marsland and J. Schaeffer, editors. *Computers, Chess, and Cognition*. Springer-Verlag, 1990.

[21] H. Matsubara. Algorithms for game programming. *Computer Software*, 10(6):3–18, 1993. (in Japanese).

[22] J. McCarthy. Chess as the Drosophila of AI. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 14, pages 227–237. Springer-Verlag, 1990.

[23] S. Minton. Constraint-based Generalization: Learning game-playing plans from single examples. In *Proceedings of the National Conference on Artificial Intelligence*, pages 251–254, Austin, TX, 1984. Morgan Kaufmann.

[24] S. Minton. A game-playing program that learns by analyzing examples. Technical Report CS-85-130, CMU, 1985.

[25] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-Based Generalization: A unifying view. *Machine Learning*, 1(1):47–80, 1986.

[26] W. Reitman and B. Wilcox. The structure and performance of the INTERIM.2 Go program. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo*, pages 711–719, 1979.

[27] C. K. Riesbeck and R. C. Schank. *Inside Case-based Reasoning*. Lawrence Erlbaum Associates, 1989.

[28] P. S. Rosenbloom. A world-championship-level Othello program. *Artificial Intelligence*, 19:279–320, 1982.

[29] Y. Saito and A. Yoshikawa. How to make stronger Go programs. *SIGAI of Information Processing Society of Japan*, (91-7):55–64, 1993. (in Japanese).

[30] Y. Saito and A. Yoshikawa. Cognitive studies for the game of Go. In *Proceedings of the Game Programming Workshop in Japan '94*, pages 44–55, 1994. (in Japanese).

[31] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, July 1959.

[32] A. L. Samuel. Some studies in machine learning using the game of checkers II – recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967.

[33] N. Sanechika et al. The specifications of "GO Generation". In *Proceedings of the Game Playing System Workshop*, pages 73–155, ICOT, Tokyo, Nov 1991.

[34] J. Schaeffer, J. Culberson, et al. A world championship caliber checkers program. *Artificial Intelligence*, 53:273–289, 1992.

[35] C. A. Seger. Implicit learning. *Psychological Bulletin*, 115(2):163–196, 1994.

[36] S. Sei, N. Sanechika, H. Oki, and T. Akaosugi. "go generation": A go playing system. Technical Report TR-0837, ICOT, 1993. (in Japanese).

[37] C. E. Shannon. Programming a computer for playing chess. *The Philosophical Magazine*, 41(314):256–275, 1950.

[38] K. Shirayanagi. Knowledge representation and its refinement in Go programs. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, chapter 18, pages 287–300. Springer-Verlag, 1990.

[39] M. H. Smith. A learning program which plays partnership dominos. *Communications of the ACM*, 16(8):462–467, August 1973.

[40] G. Tesauro. Connectionist learning of expert backgammon evaluations. In *Fifth International Conference on Machine Learning*, pages 200–206, 1988.

[41] K. VanLehn. Problem solving and cognitive skill acquisition. In M. I. Posner, editor, *Foundations of Cognitive Science*, chapter 14, pages 527–579. The MIT Press, 1990.

[42] A. Yoshikawa and Y. Saito. Cognition of board situation in Go. *SIGAI of Information Processing Society of Japan*, (91-6):41–53, 1993. (in Japanese).

[43] T. U. Zahle. A program of master-strength to play five-in-a-row. In P. A. Samet, editor, *Proceedings of Euro IFIP Congress*, pages 475–482, 1979.

Appendix

Results of world computer Go competitions

Programmer	Program	87	88	89	90	91	92	93	94
Ken Chen	Go Intellect		5	3	2	2	1	3	1
Mark Boon	Goliath	7	3	1	1	1	3		
ZhiXing Chen	Handtalk					6	2	1	3
Janusz Kraszek	Star of Poland	9	4	6	3	5	5	2	4
David Fotland	Many Faces of Go	4	8	7		10	6	4	2
Kao Kuo Yuan	Stone		7	10		7		6	5
Dong-Yue Liu	Dragon	2	2	12	4	3			
Walter & Fred Knöpfle	Modgo					8		5	6
ICOT	GOG						4		
Noriaki Sanechika	Igo		10	5	6	4			
Anders Kierulf	Explorer			4	7	13		8	
& Martin Müller									
Chung Ho Lee	Sason					9		10	7
Bruce Wilcox	Nemesis	5	11	2	5	11	7		
Yoshikawa	Dai Honinbo				9	14	8	13	
K. Hayashi	Codan		1						
Loh-Tsi Wang	Friday	1							
Kaihu Chen	Peanut	3							
Allan Scarf	Microgo 2	6	13						

Table 6.1: Results of world computer Go competitions