Flexible Acquisition of Various Types of Knowledge from Game Records: Application to the game of Go

Takuya Kojima **Kazuhiro Ueda** **Saburo Nagano**
College of Arts and Sciences
The University of Tokyo
3-8-1 Komaba, Meguro-ku, Tokyo, 153, JAPAN
kojima@taikan.c.u-tokyo.ac.jp

Abstract

A large amount of knowledge is considered very useful for systems playing games such as Go and Shogi (Japanese chess), for which the search space larger than that for chess. Knowledge acquisition is therefore necessary for systems playing these games. This paper introduces a new flexible algorithm which acquires various types of knowledge from game records. This algorithm is especially applicable to Go because Go has the largest search space among major games. The algorithm enables computer systems to acquire patterns with a variety of shapes and sizes. Sequences of moves, which are useful for human players but lacking in the present game-playing systems, are also acquired by this algorithm. The possibility of acquiring other types of knowledge — such as maxims, patterns with situations, and patterns for tsume-go — is discussed. The possibility of applying the algorithm to games other than Go is also discussed.

1 Introduction

The studies on games have mainly focused on search techniques, and these techniques enable systems playing most games — such as chess, checkers, and Othello — to perform as well as human experts. It is very difficult, however, for systems implemented using only search techniques to perform well when playing games such as Go and Shogi (Japanese chess), for which the search space is larger than that for chess. The implementation of systems playing Go as well as experts do is an especially hard problem for AI researchers. Although knowledge similar to the knowledge human experts use is expected to be useful, no systems having such knowledge exist.

The knowledge of human experts has two important features. One is that it is full of *variety*: human experts can have many types of knowledge, such as verbal and visual knowledge, and there is little limitation in size and shape. The other is that it can be of *large amount*: a chess master is estimated to have between 10,000 and 100,000 chunks of knowledge [Simon and Gilmartin, 1973].

Most computer systems which play games nowadays have much less knowledge than human experts. "Many Faces of Go" [Fotland, 1993], for example, which has more knowledge than most Go systems, has about a thousand patterns, which the programmer manages to input. Human experts, however, are said to have much more knowledge. The most effective way for systems to acquire the same amount of knowledge that human experts have may be for them to use automatic knowledge acquisition.

Although some systems have tried to acquire knowledge, most of them acquire only fixed knowledge, which is much different from what human experts have. A new flexible algorithm is therefore necessary.

Knowledge generally needs to be acquired from sources, and the game records of experts are appropriate sources of game knowledge for two reasons: they are easy to get and they are reliable. Since games between experts in most widely played games are often recorded, the game records are easy to get. And since most moves of human experts are almost perfect (less than a few percent are bad moves), the game records are very reliable.

The work described here therefore aims to acquire a large amount of knowledge with a variety of sizes and shapes, which are similar to those human players have, from game records of expert players. We chose an evolutionary approach to acquire knowledge. Acquired knowledge can be useful not only for computer Go systems but also for human players because it is said to be very good for players to know good patterns.

In order to acquire useful knowledge with a variety of shapes and sizes, a new evolutionary algorithm is introduced. In ordinary evolutionary algorithms, individuals tend to become similar and converge at one point. In this new algorithm, this convergence is avoided by introducing an activation value to individuals and by introducing a new variation-making operator, splitting. Both of these innovations are inspired by ecological systems.

2 Classification of Go Knowledge

We classify Go knowledge according to two criteria: form and the degree of validity. First we classify it into

three types according to the form: patterns, sequences of moves, and maxims. A *pattern* is a rule whose condition part is a part of the board configuration and whose action part is a move, a *sequence of moves* is a rule which suggests several moves, and a *maxim* is a rule consisting of Go terms. We then classify Go knowledge into two types according to its degree of validity: strict knowledge and heuristic knowledge. *Strict* knowledge is knowledge that can always be applied and whose validity can be shown easily. For example, in *tsume-go*, or life and death problems, why and how stones are killed can be shown. *Heuristic* knowledge, on the other hand, is knowledge whose effectiveness is hard to show and whose validity depends on situations. For example, the maxim "When opponent's move is *tsuke*, then play *hane*" is not always true and sometimes it is better to play *nobi* when the opponent's move is *tsuke*. In Go, a search space is so huge and moves are so sensitive to situations that most of the knowledge used by human players is heuristic knowledge.

Most previous studies focused on patterns. Our previous research [Kojima *et al.*, 1994; Kojima, 1995] focused on strict patterns and introduced an algorithm that deductively acquires patterns used in capturing stones. The algorithm is given examples of human players in capturing stones and explains why they can capture stones. In this process, it detects important stones or edges for capturing the stones and extracts them from the whole board. As a result, it acquires patterns in capturing stones. Although it acquires reliable patterns, it can only acquire strict rules, not heuristic ones.

Some studies focus on heuristic rules. One system [Sei and Kawashima, 1994], for example, collects from master's game records patterns, which are fixed to form the shape of diamond, and whose area is within five Manhattan distances of the center. There are, however, two problems in this system: one is that the shape of patterns is fixed and the other is that the size of patterns is limited. All patterns known to be useful are not diamond-shaped and there are many patterns which require wider area than those stored in the system.

This paper introduces a flexible algorithm which acquires various types of heuristic knowledge that human experts have.

3 Algorithm

We have implemented a new evolutionary algorithm extended by ecological analogy, which enables various types of knowledge to be acquired flexibly from game records. Although this algorithm can be applied to various games, this paper explains only the modified procedure for acquiring patterns and sequences of moves of Go. For the detailed and more general explanation of the algorithm, see [Kojima *et al.*, 1997].

3.1 Overview of the Algorithm

Since this algorithm is inspired by ecological systems, it is explained here using ecological terms. Each rule is considered as an *individual* and has an *activation value*. There are no rules in the initial state. A training datum is considered as *food* which is eaten by any rule that matches it, and the activation value of the rule increases. If there are no rules which match a given datum, a new rule that matches the datum is created. Thus the number of rules increases in an early stage. Rules whose activation value exceeds a threshold value *split* into the original rule and a more specific rule. Every rule eats food during each step and decreases one activation value. Rules whose activation value is 0 *die*. The procedure of this algorithm is shown below.

Algorithm 1 Outline of this Algorithm

```
1   step ← 1
    while step ≤ laststep
2       choose a game record from a game database
3       move ← 1
        while move ≤ lastmove
4           a training datum ← moveth move
5           if no rule matches the datum
                then create a new rule
                else feed matched rules
            for all rules
6               if activation of a rule > {threshold}
                    then split the rule
7               activation of a rule ← activation of a rule − 1
8               if activation of a rule = 0
                    then the rule dies
            end for
9           move ← move + 1
        end while
10      step ← step +1
    end while
```

3.2 Details of the algorithm

Rules

A rule takes the form of a production rule consisting of a condition part and an action part. It is described in relative terms so that it can be executed by either player. A condition part consists of predicates, which are pairs of an object and its coordinates (*coordinate=object*). The action point is the point on which a stone is placed and has the coordinate [0,0]. Other coordinates are relative to the action point, and objects represent either stones or edges of the board. Stones are either "SAME" (i.e. belonging to the active player), "DIFF" (belonging to the opponent), or a minus integer, which represents the previous move.[1] The action part of a rule is always "place a stone to [0,0]", and therefore is omitted in the following examples.

For example, "**If** $[1,0] = -1 \wedge [2,1] =$ SAME $\wedge [0,1] =$ DIFF $\wedge [5,0] =$ EDGE" means that **if** the previous move is played in point [1,0] (relative to the action point) and a stone *same* the color as the active player's exists in point [2,1] and a stone of a *different* color exists in point [0,1] and an *edge* exists in point [5,0], **then** put a stone on [0,0].

[1] "−3" represents the move played three moves before. This is used to acquire sequences of moves.

Matching and Feeding Rules

Matched rules are those whose condition part and action part match a given training datum. Those rules that do not have rules more specific than themselves are fed. An activation value equal to Parameter $FOOD$ is shared among the rules fed by one training datum. The following is an example.

Suppose that the following five rules are matched.

1. **IF** C_1 **THEN** A_1
2. **IF** $C_1 \wedge C_2$ **THEN** A_1
3. **IF** $C_2 \wedge C_3$ **THEN** A_1
4. **IF** $C_4 \wedge C_5$ **THEN** A_1
5. **IF** $C_2 \wedge C_3 \wedge C_4$ **THEN** A_1

Since Rules 1 and 3 have more specific rules than themselves (Rules 2 and 5, respectively), they are not fed. The others (Rules 2, 4, and 5) share food and each gets one third of the food.

In summary, the more general the rules are, the more frequently they are matched, but the lower the probability that they are fed when they are matched.

Splitting and Creating Rules

A rule whose activation value is greater than a certain threshold value splits into two rules: one is the original rule and the other is created by adding a new predicate to the condition part of the original (parent) rule. For example, a pattern, **IF** $C_1 \wedge C_2$ **THEN** A_1, splits into the original rule and the new pattern, **IF** $C_1 \wedge C_2 \wedge C_3$ **THEN** A_1. The new predicate, C_3, is randomly chosen as long as the newly created rule matches the given datum.

If no rule is matched, a new rule is created which has only one predicate chosen randomly in the condition part as long as the newly created rule matches the given datum.

A newly created rule that is the same as a rule already in the rule set is deleted. Thus there is no duplication of rules.

In splitting a rule (called a target rule) and creating a new rule, a new predicate is chosen among the stones on the board, the edges of the board, or the previous move. The previous move is selected as follows. As mentioned before, a minus integer in the object part of the condition represents a previous move. For example "-3" represents a move played three moves before the action move. If the target rule has no previous move in the condition part, the move just before the action move (-1) is added to the candidate conditions that can be chosen. If the target rule has previous moves in the condition part, the move just before the move in the condition is added to the candidate conditions. For example, if $-2 and -1$ are in the target rule, -3 is added to the candidate conditions.

Parameters

This algorithm has two parameters: one is a constant, INI_ACT, which is the initial activation value for newly created rules; the other is $FOOD$, which is a function of the number of games used as training data (i) and which is the total amount of activation given by one training datum.

In an experiment with 9x9 board games, the parameters were defined as follows: INI_ACT is 200, and if $i < 4000$, $FOOD = 400$; if $i \geq 4000$, $FOOD = i \div 10$. In an experiment with 19x19 games, the parameters are defined as follows: INI_ACT is 1000, and if $i < 4000$, $FOOD = 2000$; if $i \geq 4000$, $FOOD = i \div 2$.

The threshold value for splitting is twice the value of $FOOD$.

Parameter INI_ACT is smaller than $FOOD$. A newly created rule is an assumption. Thus it is often inappropriate and consequently dies, since a new condition is chosen randomly. Therefore, the activation value given to a newly created rule is smaller than that given to an existing rule.

The number of iterations is counted by the number of games used. It is 50,000 for 9x9 board games and is 26,500 for 19x19 board games.

3.3 General Description of the System Behavior

In the initial state, this system has no rules. A training datum is given and if no rules match the given datum, a new rule having only one predicate in the condition part is created. Thus many rules with only one condition are created in the early stages. A rule whose activation value is over the threshold value splits into the original rule and a more specific rule created by adding a new predicate to the condition, so rules that often match data split frequently and more specific rules are created. The frequency of matching for a newly created rule is always less or equal to that for its parent rule, because the condition of the new rule is always more specific than that of its parent. Rules split and evolve into more specific rules as long as the frequency of being fed is higher than a certain threshold frequency. Therefore, in the stable state all rules are expected to match data with almost the same frequency.

The number of rules is expected to be almost equal to the amount of $FOOD$. Let N be the number of rules. In each step, every rule loses one unit from its activation value, so the total loss in activation value is N; on the other hand, the total gain in activation value is $FOOD$.

4 Application to the Game of Go

We chose the game of Go as a testbed for the algorithm, because Go has a much larger search space than chess and knowledge is therefore indispensable for computer systems to play Go. Moreover, recent cognitive studies on Go players [Saito and Yoshikawa, 1996] have revealed that human experts use Go knowledge such as "patterns" and also search a game tree a little. This shows the importance of Go knowledge to human players. Studying the acquisition of Go knowledge may therefore help Go playing systems play better.

Patterns and sequences of moves are acquired, and the possibility of acquiring patterns with situations, maxims, and patterns of tsume-go is discussed in this section.

4.1 Patterns Acquired by the Previous System

In the previous version of the system[Kojima *et al.*, 1996], the opponent move just before the action move has the coordinate [0,0], thus the two sequences of moves are always acquired.

Rules are acquired using 9x9-board-game records. Of the 4,121 acquired rules, the top 20 rules (the order being determined by the number of being fed) are evaluated by an expert. There are three kinds of evaluation.

Firstly, the expert is shown a condition part of a rule and shows some candidate moves for action (average 3.2 moves). Among 20 rules, 18 rules are matched with one of the candidate moves the expert shows. Another rule are not matched, but it is a typical pattern of 9x9 board games and the expert who is familiar with 19x19 board games takes it strange.

Secondly, the expert evaluates whether the board configuration of a condition part of a rule is familiar to him. It is evaluated by three categories, bad, average, or good. 14 rules are evaluated by good and 4 average and 2 bad.

Lastly, the expert evaluates whether an action move of a rule is familiar to him. It is evaluated by three categories, bad, average, or good. 15 rules are evaluated by good and 4 average and 1 bad, which is typical of 9x9 board games explained above.

From the three evaluations, most rules acquired are evaluated good or average, so it can be said that they are appropriate.

Although the results are very good, only one kind of knowledge containing two sequences of moves are acquired and other knowledge — such as patterns without the previous opponent move or sequences of moves suggesting more than two moves — can not be acquired, because the coordinate [0,0] is the previous opponent move. Therefore the coordinate [0,0] is the action point in this system; this enables the system to acquire other knowledge.

4.2 Patterns Acquired from 9x9 Board Games

Of 4,684 acquired rules, those which are fed over 100 times were selected for evaluation, because rules that are just created may be inappropriate and die after only a few steps. Of the 3,211 selected rules, the top 5% (161 rules) (the order being determined by the number of being fed) were evaluated. Moves indicated by the rules were evaluated and categorized as good, average, or bad. Of the 161 rules, 67 (41.6%) were evaluated as good, 34 (21.1%) as average, and 60 (37.3%) as bad.

4.3 Patterns Acquired from 19x19 Board Games

Rules are acquired using 19x19-board-game records by the modification to this algorithm.

Vision

The direct application of this algorithm to 19x19 board games fails to acquire good patterns because the search space is much larger than that of 9x9 board games. Therefore we introduce a concept, *vision*[2], a domain specific bias to narrow the search space. This concept is based on the idea that when a stone is placed according to a rule, all the objects in the condition part should be *visible* from the placed stone.

When a rule is used, objects in the condition part of the rule become transparent.

One stone is visible from another stone, if and only if there is nothing opaque in the region surrounded by both of the stones and two parallel lines touching to both of the stones.

Take Figure 1 for example. From ❶, only Stone **a** is visible. When a rule containing **a** in the condition is used, Stones **b,c**, and **d** are visible from ❶.

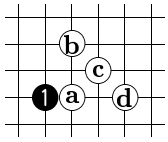


Figure 1: An explanation of the concept of vision.

This concept is used in two cases: one is when rules are split and the other is when rules are matched. When rules are split, a new condition is chosen. Without vision, a new condition is randomly chosen among all stones and edges of the board. The number of the candidate conditions is very large when 19x19 board games are used. With vision, conditions which may have little to do with the rule are excluded and only visible stones are considered. Thus the search space is narrowed.

When rules are matched, the only rules considered are those whose condition parts are all visible from the placed stone. Without vision, only stones and edges in the condition part are considered and empty spaces between objects are not taken into consideration. With vision, empty spaces between objects are implicitly considered because the space between objects in the condition and the placed stone should be empty.

This concept is very simple and natural and may be applied to other board games.

Results of acquisition

2,734 rules (both patterns and sequences of moves) were acquired. Twenty-seven rules (1% of acquired rules) were randomly selected and evaluated by an expert, who classified them into three categories: good, average, and strange. Among the 27 rules, 14 (52%) were evaluated as good, 6 (22%) as average, and 7 (26%) as strange.

Among 120872 moves, 120861 (99.99%) could be explained by acquired rules.

Examples of acquired rules

In previous studies, only small and fixed-shaped rules were acquired, whereas this algorithm permits the rules to be any size and shape.

[2]The implemented function is called *visible*.

Our algorithm acquires rules with a variety of the sizes, never realized in previous studies. Figure 2[3] is an example of a large (whole-board) rule. Figure 3[4] is an example of a small rule.

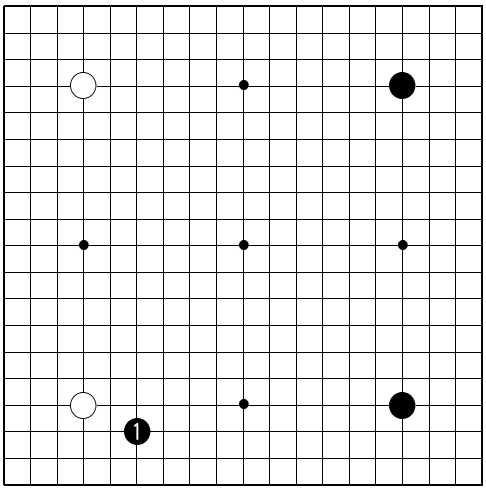


Figure 2: An example of a large size rule

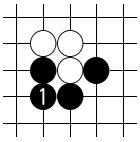


Figure 3: An example of a small size rule

The variety in shape of the rules acquired by our algorithm is demonstrated by the following three examples. Figure 2 is an example of the rule with stones spread over a wide area. Figure 3 is an example of a compact, square rule. Figure 4 is an example of an irregularly shaped rule.

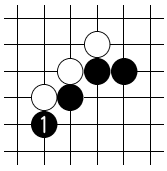


Figure 4: An example of an irregular shape

The examples show the great variety in size, shape, and context of knowledge acquired by this algorithm.

[3]If [10,1]=SAME [10,13]=SAME [-2,1]=DIFF [-2,13]=DIFF [0,-3]=EDGE [-6,0]=EDGE

[4]If [1,0]=SAME [0,1]=SAME [2,1]=SAME [1,1]=DIFF [0,2]=DIFF [1,2]=DIFF

4.4 Sequences of Moves

Sequences of moves, which suggest several moves, are very useful for human players but are lacking in the present computer systems [Saito, 1996]. They may thus be very useful for helping computers to play better. They are acquired by adding previous moves in splitting a rule as mentioned above.

Figure 5 is an example of sequences of moves acquired by this algorithm.

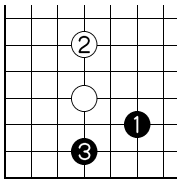


Figure 5: An example of sequences of moves.

4.5 Maxims

Maxims are considered as very useful knowledge representation both for computers and for human players to play Go. [Shirayanagi, 1990] is the most famous system using maxims to play Go, considered "the most promising but at the same time the most difficult to implement" [Erbach, 1992]. This is considered "the most difficult to implement" because maxims themselves must be hand-coded by the programmers using Go terms defined by Shirayanagi. Our algorithm may acquire Go maxims using Go terms, which has already defined by Shirayanagi. The reason is as follows. Rules of our algorithm are Horn clauses, so predicates of rules can be Go terms. In splitting, a new predicate, which is a Go term, is added to a rule condition and the rule becomes more complex. In this way, many rules using Go terms, which we call *maxims*, will be acquired by our algorithm.

4.6 Patterns with situations

Go playing systems nowadays are said not to behave differently according to situations they face and said that this is their weakest point. Overcoming the weakness is very difficult because of the difficulty of writing so many rules according to situations. Once situation terms in Go are defined, Go patterns or rules with situations may be acquired using the terms, just as Go maxims are acquired. This may enable computer Go to be much stronger.

4.7 Tsume-go

Tsume-go is life and death problems. Cognitive studies of experts solving tsume-go show that they use knowledge or patterns[Yoshikawa and Saito, 1996]. Knowledge acquisition from tsume-go problems may therefore be very useful both for human players and for computers. Since tsume-go problems can be considered as game records of a few moves (usually one or three moves), this algorithm can be applied to tsume-go.

5 Discussion

5.1 Application to Other Games

We expect that this algorithm can be applied to other games than Go for the following three reasons. The first is that this algorithm is flexible; rules of this algorithm are Horn clauses, which are very flexible, and most rules used in games may be described by Horn clauses. The second reason is that sources of knowledge are game records, which for most games are easy to get. And the third reason is that this algorithm uses no knowledge of games.

5.2 Improvement of the results

The human expert's evaluation of the acquired rules was shown above. Most rules evaluated as average or bad lacked some conditions, or stones. The complexity of the rules depends on the number of iterations; the more the algorithm iterates, the more specific the rules are. The number of the rules acquired depends on the parameter *FOOD*. Therefore, changing the parameter and the number of iterations may improve the quality of the acquired rules. We are currently studying this assumption.

6 Conclusions

A new algorithm for acquiring various types of Go knowledge from game records is introduced. Patterns are acquired from 9x9 board-game records and 19x19 board-game records. Although previous studies that use game records store only fixed-shaped patterns, this algorithm acquires patterns with a large variety of shapes and sizes. Sequences of moves, which are regarded as important for Go players, are also acquired. Acquired patterns are evaluated by a human Go expert and most rules extracted from 9x9 board games and many rules extracted from 19x19 board games are evaluated as appropriate.

The possibility of acquiring other types of rules — such as maxims, patterns with situations, and patterns for tsume-go — is discussed.

This algorithm is very flexible, and the possibility of applying it to other games than Go is also discussed.

Acknowledgements

The authors would like to thank Mr. Masahiro Okasaki for useful information and to thank Masahiro Hachimori and Kiyoshi Izumi for their helpful comments.

References

[Erbach, 1992] David W. Erbach. Computers and Go. In R. Bozulich, editor, *The Go Player's Almanac*, chapter 11, pages 205–217. Ishi Press, 1992.

[Fotland, 1993] David Fotland. Knowledge representation in the Many Faces of Go. Manuscript available by Internet anonymous ftp from `basderver.ucsf.edu:Go/comp/mfg.Z`, 1993.

[Kojima *et al.*, 1994] Takuya Kojima, Kazuhiro Ueda, and Saburo 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.

[Kojima *et al.*, 1996] Takuya Kojima, Kazuhiro Ueda, and Saburo Nagano. A case study on acquisition of pattern knowledge in Go using ecological analogy. In *Game Programming Workshop in Japan '96*, pages 133–140, 1996. in Japanese.

[Kojima *et al.*, 1997] Takuya Kojima, Kazuhiro Ueda, and Saburo Nagano. An evolutionary algorithm extended by ecological analogy and its application to the game of Go. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[Kojima, 1995] Takuya Kojima. A model of acquisition and refinement of deductive rules in the game of Go. Master's thesis, The University of Tokyo, 1995.

[Saito and Yoshikawa, 1996] Yasuki Saito and Atsushi Yoshikawa. An analysis of strong go-players' protocols. In *Game Programming Workshop in Japan '96*, pages 66–75, 1996.

[Saito, 1996] Yasuki Saito. *Cognitive Scientific Study of Go.* PhD thesis, The University of Tokyo, 1996. in Japanese.

[Sei and Kawashima, 1994] Shinichi Sei and Toshiaki Kawashima. The experiment of creating move from "local pattern" knowledge in Go program. In *Game Programming Workshop in Japan '94*, pages 97–104, 1994. in Japanese.

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

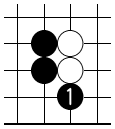
[Simon and Gilmartin, 1973] Herbert A. Simon and Kevin Gilmartin. A simulation of memory for chess positions. *Cognitive Psychology*, 5:29–46, 1973.

[Yoshikawa and Saito, 1996] Atsushi Yoshikawa and Yasuki Saito. Can not solve Tsume-Go problems without looking ahead? In *Game Programming Workshop in Japan '96*, pages 76–83, 1996. in Japanese.

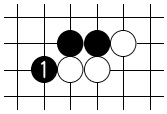
Appendix: Examples of Acquired Rules

The following are examples of acquired rules in 19x19 board games.

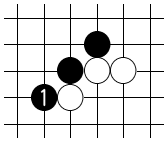
1.



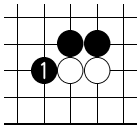
2.



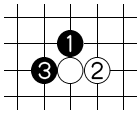
3.



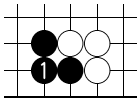
4.



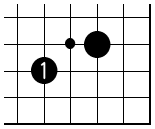
5.



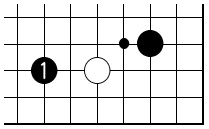
6.



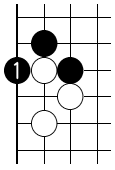
7.



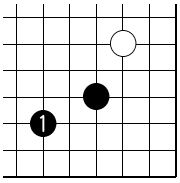
8.



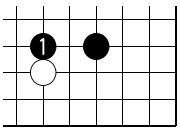
9.



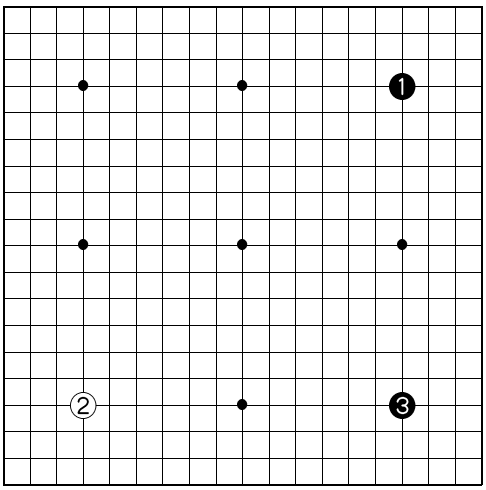
10.



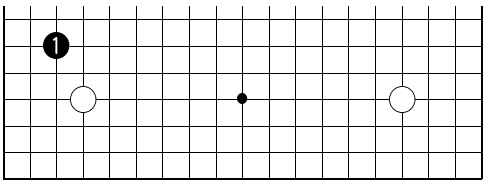
11.



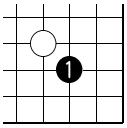
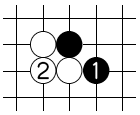
12.



13.



14.



15.