A positional Judgment System for Computer Go

Shi-Jim Yen, Shun-Chin Hsu
Department of Computer Science and Information Engineering of National Dong Hwa University, Taiwan
sjyen@mail.ndhu.edu.tw

Abstract

Computer Go offers researchers a new challenge and opens up a very wide scope of possibilities for artificial intelligence. In a computer Go program, the most important element is a positional judgment system. Following the methods of human Go experts, we designed and implemented a new model of positional judgment for computer Go. This model was employed successfully in a computer Go program, Jimmy-5.0, which beat the latest world-champion Go program, ManyFaces, in the 1998 FOST Go contest.

Keywords: Computer Go, Artificial Intelligence, Positional Judgment.

1 Introduction

Despite the simplicity of the rules, Go is far more complicated than other board games respective to computer [Allis et. al., 1991]. 10**40 different board configurations were estimated for Chess by A. Newell [Newell et. al., 1958]. But compared to the complexity of Go, these numbers are insignificant. Ignoring complications produced by the removal of stones, the number of legal sequences of stone placement in a Go game is 361! - A value which reaches about 10**761. Exhaustive searching of such a large game tree is an impossible task even for the fastest computer in the world. In fact, no two games with an identical sequence of moves have ever been played through out the entire history of Go [Hsu et. al., 1994].

Go is a two-player board game. Two players alternate places a black and a white stone on some empty intersection of a 19 by 19 grid. Stones are never moved, and only removed, called captured or killed, if they are completely surrounded. The objective of Go game is to secure more territory than the opponent. For more detailed description of the game and the rules of Go, readers can visit American Go Association Home Page: http://www.usgo.org/.

In computer chess, a program can play at the master level by applying a few simple state space search algorithms and some complex heuristic functions [Berliner, 1974]. However, this does not work very well with Go. One of the reasons is that a quick and fine evaluation function is difficult to develop. To solve the problem is the main topic of this paper.

Since brute force search can not work in computer Go, knowledge is expected to play a major role in this domain. Computer Go programming often involves a problem of simulation of human vision and perception [Reitman and Wilcox, 1978]. Human master of Go usually use “feeling” or sense as a guide to extract important information from the spatial configuration of stones on the board before making their moves. However, it is not easy to simulate such a procedure in computer Go programming. In order to solve this problem, many data structures for simulating human’s strategies was developed. Base on these data structures, a positional judgment system was constructed.

2. Basic data structure and basic tools

A good computer Go program should have the ability to know the status of a game. This includes recognition of stones, *strings*, *links*, *groups* and *influences*. A string consists of identical color and directly adjacent stones. When a string is graphically associated with other identical color strings, we say that there is a link between them. The link relationship between strings is like the equality relationship in mathematics. For example, i.e., if string *a* links to string *b*, and string *b* links to string *c*, then string *a* links to string *c*. When strings with link relationship to each other form a group; if there is no link around a string, the string forms a group itself. Finally, every group radiates influences across the board. The values of the influences depend on their type or status. The purpose of influence structure is to represent the concept of thickness.

Figure 1 shows the relationship among their structures. In Figure 2, Black stone 1 and stone 7 form a string. Since there is a link denoted by "X" between this string and stone 5, they form a group. Meanwhile, white stone 4 and stone 8 form a string. Stone 6, stone 2 and this string form a group by the same reason. Figure 3 shows the influences of each stone on the board of figure 2.

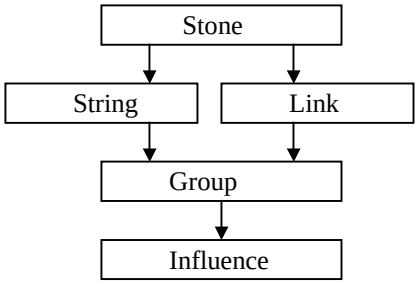

Fig. 1. Data structure relationship.

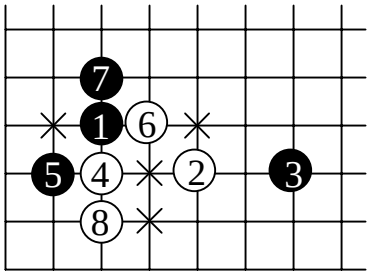

Fig. 2. An example of data structure.

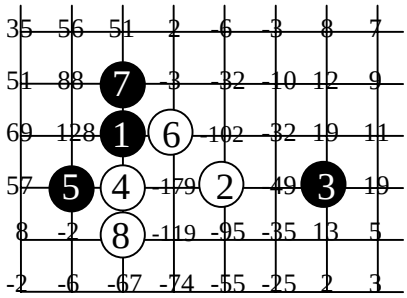

Fig. 3. An example of influences.

There are three situations for a group: *alive*, *dead* or *critical*. A group is alive if we can always prevent it from being captured no matter how the opponent, playing first, attacks. A group is dead if the opponent can always capture it, no matter how we, playing first, defend. A group is critical it will be alive if we move first and it will be dead if the opponent moves first.

The main corresponding attributes of those data structures are shown as follows.

(1) **Point**:
- (a) Status: White, black or empty.
- (b) Position: The horizontal and the vertical show the stone's position.

(2) **String**:
- (a) Color: White or black.
- (b) Situation: Alive, dead or critical.
- (c) Number of stones.

(3) **Link**:
- (a) Status: If status is true, then all the strings with the same color around the link position belong to same group.

(4) **Group**:
- (a) Color: White or black.
- (b) Territory: The territories occupied by a group.
- (c) Situation: Alive, dead or critical.
- (d) Outlet: The outlets of this group.

(5) **Influence:**
- (a) Strength: Integer number, which is used to represent the strength affected by each group on the board.

Two important tools are used in our system. One is a pattern match system; the other is a string capture system. A pattern match system can recognize a special pattern/shape and a string capture system can recognize whether a string is dead or not. The details of the two systems have been discussed in [Hsu et. al., 1994][Hsu and Liu, 1991][Kojima et. al., 1996][Lorentz, 1995][Müller, 1995].

3. Requirements of a positional judgment system

A positional judgment system is the basic element of a computer chess program. There are many simple and precise evaluation methods for computer chess. But it seems difficult to analyze a Go game by a computer program. Since the traditional methods for computer chess could not work well in Go, new methods must be developed. In this paper, we try to simulate the method used by human Go experts.

Many theories of position judgment by human Go experts are discussed, such as in [Cho, 1989]. Three key elements must be considered: *territory*, *thickness* and *moyo*. First, estimations of definite territory are required as a guide at each stage of a game. Next, we should count territory during the early stages or during the fighting stage when things are still unclear. One-question concerns strong outward facing shapes which are equivalent to territory and form thickness or

influence. Thickness of course cannot be called definite territory. However, we must take such potential territory into account or the analysis would be incomplete. Thus, we can induct three main points for positional judgment: counting definite territory, counting thickness and counting moyo [Cho, 1989].

Figure 4 shows the relationship between our data structures and the three main points. From the group's information/attributes, we calculate the definite territory by adding each group's territories. For counting thickness, first we should know the status of each group, then consider their influences. Influences can also help to estimate a moyo. By the above discussion, we know that group information and influences are the basic information for a positional judgment system. If the precise values of them are found, it is easy to analyze a Go game. Therefore, we focus on how to recognize and quantify the group information and influences in this paper.

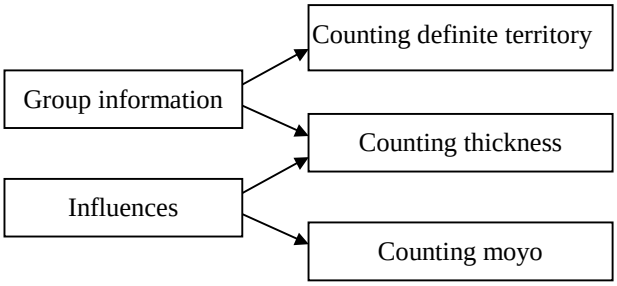

Fig. 4. The relation ship between data structures and three main points of game analysis.

4. The system

4.1 System configuration

The basic point of position judgment for Go is identification of groups. Group identification in computer Go is a difficult problem. In this paper, we develop a new method by using strict definitions on links and influences. Two subsystems are used to recognize links. The links and rough influences identify groups. Then we can rough analyze the information/attributes of each group, such as territories, outlets, and status. For a group with a lot of territories, we know it is alive. For a group with outlet and few territories, we use a searching subsystem to judge if it is dead. In the first rough analysis of each group, we can recognize stable groups (alive or dead). That information can help us to analyze those unstable groups (critical) in the second round. By the information of all groups, more strict influences are constructed. Finally, our system's outputs are constructed. The system configuration is shown in figure 5. The Input of this system is a sequence of moves of a Go game. The Outputs are group information and final Influence.

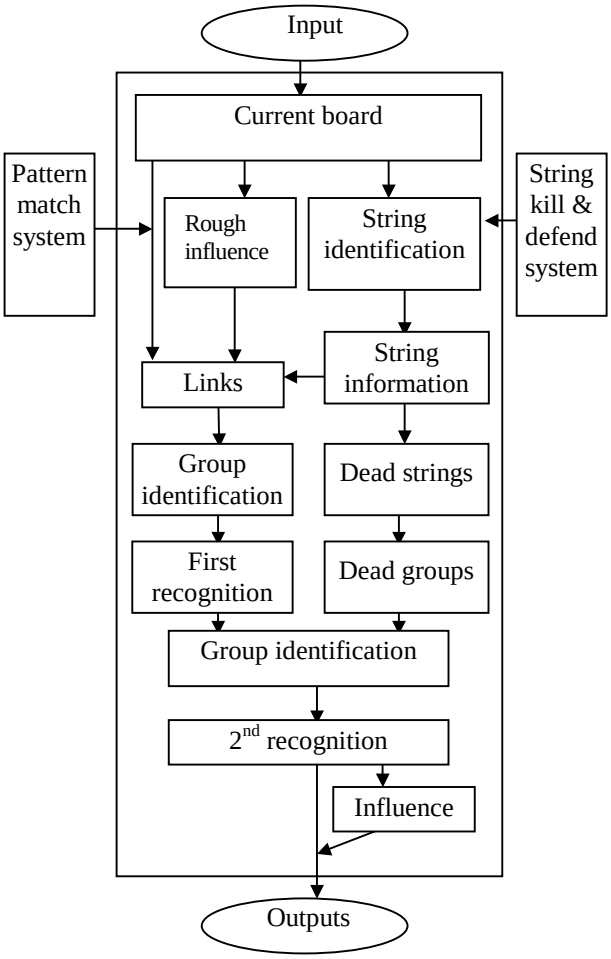

Fig. 5. The system configuration.

4.2 Rough influence

The concept of influence has been used many times in computer Go. The reason is that influence can simulate human player's vision estimation of thickness. Basic ideas of potential influence are described as follows. Every stone on the board radiates influences across the board. The influence value is maximum at its immediate neighboring points and decays as distance increases. The black (resp. white) influence is the sum of influence of each black (resp. white) stone. Total influence of a game is the difference of black influence and white influence.

Some theories about influence were also discussed by human Go experts [Cho, 1989]. There is a common sense of thickness. The influence of a single stone is about two lines and two connected

stone can influence three line spaces. Therefore, three stones connected side by side have influence of four lines. Figure 6 shows an example of the influence theory.

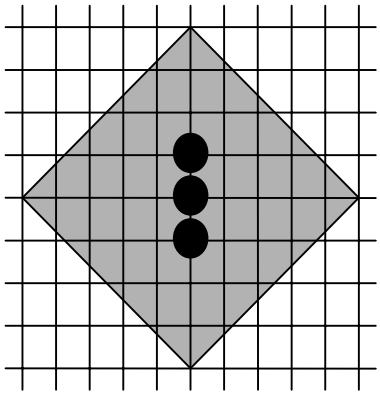

Fig. 6. An example of the influence theory.

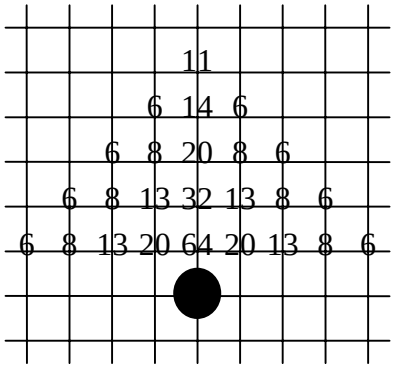

Fig. 7. The potential influence table.

We will develop a new method to evaluate more precise influence based on those ideas. First, a potential influence table is constructed in figure 7. This table constructs our rough influence with the following method. When the system works on some stone, the stone checks its adjacent four neighbors: up, down, right and left. If there is no stone in some direction, it adds the value in that direction (respective to the table) to the rough influence. Figure 8 shows the rough influence of a single black stone. Figure 9 shows another example. Since two white stones obstruct the stone, the influence of the black stone in figure 9 is weaker than the single stone in figure 8.

Figure 10 shows the rough influence of the example in figure 6. In order to apply the common sense of thickness, we say that a point is under black's (resp. white) influence when its value is over 25 (resp. below -25), otherwise, it is under black's (resp. white) control if its value is over 100 (resp. below -100).

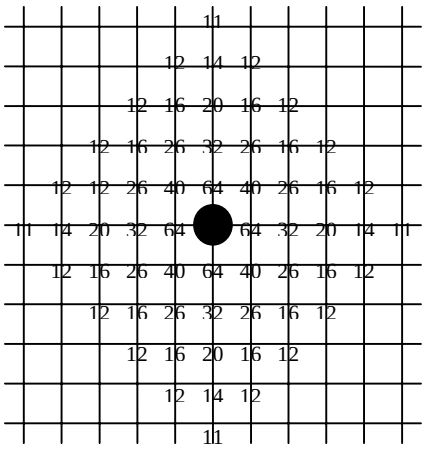

Fig. 8. Rough influence of a single stone.

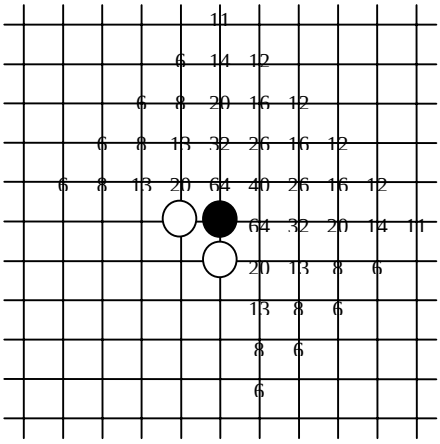

Fig. 9. Rough influence of the black stone.

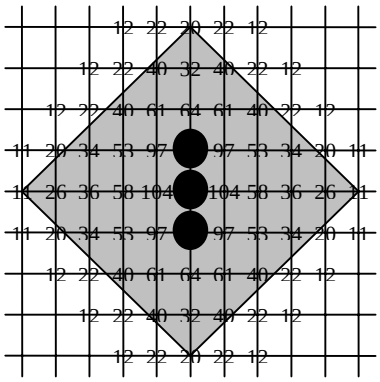

Fig. 10. Rough influence of three connected black stones.

4.3 **Link**

We say that there is a link between two strings if the two strings have the same color and the opponent can not disconnect them or the opponent disconnect them will get no benefit. Intuitively, search can get a better result for recognizing link. But it will cause system load overhead. Therefore, we try to recognize link by minimum search. The link conditions are classified into three cases. By solving those cases, almost any link could be recognized. Those cases are stated and analyzed as follows.

(a) When check some position, the position and near positions match some pattern. Pattern match can recognize most links. We create and correct many link patterns by experience knowledge. In our system, we build a database including about 100 patterns. From our experiences, pattern match can recognize about 80% links correctly. Figure 11 and figure 12 show some link pattern examples.

(b) There is no opponent stone near the position and there is a very heavy influence value in the position. This case could replenish case a. Since the position has no stone around the position and the position has a very heavy influence value, if the opponent try to cut those strings, the opponent will get no benefit. Therefore, there is a link in this position.

(c) There is an opponent's dead string in the position. When an opponent string breaks two strings with same color, it is hard to recognize the link between the two strings by pattern match. In order to solve this situation, our system use the opponent string information constructed by string capture system and apply the following rule: if some opponent string is dead, then each stone of the string forms a link of its adjacent neighboring opponent strings.

Our system recognizes links based on the three cases. After testing more than 500 games, the system makes mistakes about 2 %. (In a game with about 250 moves, it makes mistakes about 5 times.) The error rate achieved by the system is similar to that of a human player with 3-kyu. This result is sufficient to construct a good Go program.

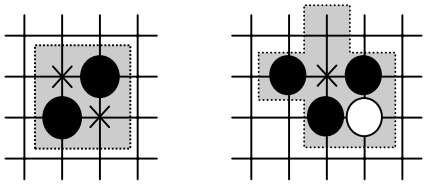

Fig. 11. Some link patterns.

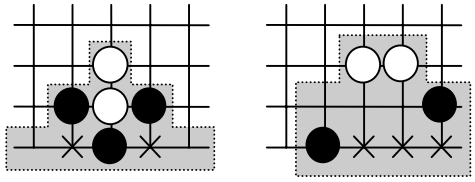

Fig. 12. Some link patterns.

4.4 **Group identification**

When each link is recognized, it is easy to identify groups. If there is a link between two strings with the same color, the two strings will belong to the same group. Since the link relationship between two strings is similar to the equality relation in mathematics, we only need to scan each string once and use a set equality algorithm [Brassard and Bratley, 1988], then each group can be identified.

Our methods for identifying groups are similiar to those proposed by [Chen, 1989][Ryder 1971][Sanechika 1991][Zobrist 1969]. However, they can identify groups more correctly by the following reasons. First, by using influence function, patterns and string information, our method can recognize links more correctly. Second, we identify groups iteratively. Therefore, if two groups with the same color are disconnected by an opponent group which is dead, the two groups will be combined into one group.

4.5 **Group information**

The most important information of group is the size of each group's territories. Figure 13 shows two examples of the formed territories of a group. The land, link, and edge-link are represented by square, "X", and "D". (An edge-link is formed when a stone is near to the edge and the around positions match some pattern.) We can find that the territories are always closed and formed by links and edge-links. Therefore, if we can recognize links and edge-links, it is easy to calculate a group's territories. The method for recognizing an edge-link is simulating to recognizing a link. Figure 14 shows an example of an edge-link pattern.

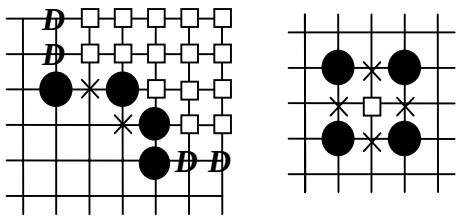

Fig. 13. The formed territories of a group.

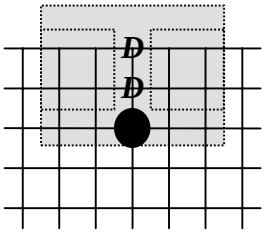

Fig. 14. An edge-link pattern.

The other important information of a group is the status of each group. For a group with few territories, we detect the "eyes" of the group. If there are no enough eyes, we test the closure of the group. First, we consider if the group can connect to some live groups. Then we consider the outlets of the group. There are some methods for calculating the outlets of a group. [Hwang and Hsu, 1994][Hsu and Liu, 1991]. In our system, we calculate them by using rough influence and a lot of patterns. If rough influence of the group is heavy or more than two outlet patterns around the group have been found, the group is alive, otherwise, it is an unstable group. For an unstable group, if opponent's living groups surround it, then it is dead.

4.6 **Final influence**

Since the strength of each stone on the board is not the same. We revise the rough influence after the status of each group is found. If a group is alive, its final influence is heavier. Otherwise if a group is dead, there is no final influence of it. For an unstable group, its final influence is weak.

5 The results

In this paper, we have described a new model for positional judgment for computer Go. We also suggest some new heuristic methods for recognizing links and influences. The system has been used successfully in a computer Go program Jimmy 5.0 which is one of the best Go programs in the world. It beat the new world-champion Go program, ManyFaces, in the 1998 FOST Go game. It took about two years to implement these ideas in our program. The size of related codes is about 5000 lines. We tested the system by playing with human players or other computer programs about 500 games in a year. About 80% of the game analysis is similar to a good human player. Despite the judgment of the degree of an unstable group, the position judgment of the system is almost the same as that of a good human player. The result is satisfactory. But for an unstable group, the system may make mistakes in the estimation of its status. The most important reason is that we only use local search to the groups. A good human player can use global search to judge the status of an unstable group and get a better result.

In order to test our system, we also choose a lot of games played by human Go experts and analyze them by the system. Compare to human player's positional judgment, the result is satisfactory, too. Figure 15 and figure 16 show an example from the 7th Meijin title match [Cho, 1989]. The players are Otake 9-dan and Cho 9-dan. In figure 15, Cho says that Black makes mistake when entering at the 3-3 point in the lower left corner. White leads Black in this time. The result analyze by our system is described as follows. The definite territories of White are 40 points and the definite territories of Black are 53 points. Except for 5.5 komi, Black lead about 7.5 points in definite territory. But from the final influence values of our system, we can find that 82 points are under influence by White and 50 points are under influence by Black. White lead Black 32 points here. Note that those points, which are under influence by someone, will increase his definite territory. Thus, the system outputs that White slightly leads Black. This result is the same as the judgment by Cho.

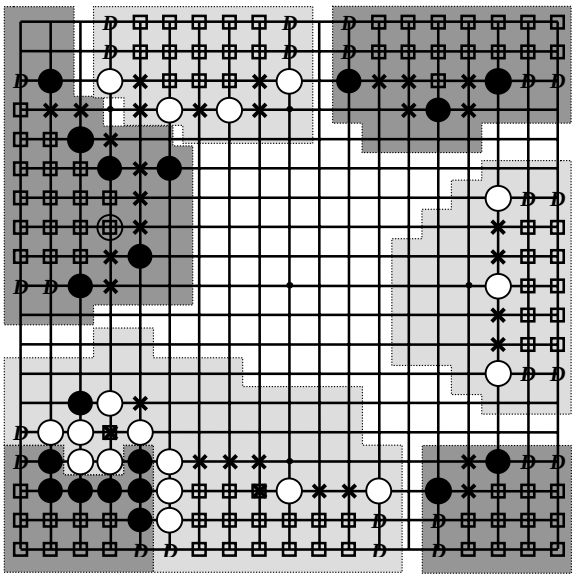

Fig. 15. The 7th Meijin title match (1).

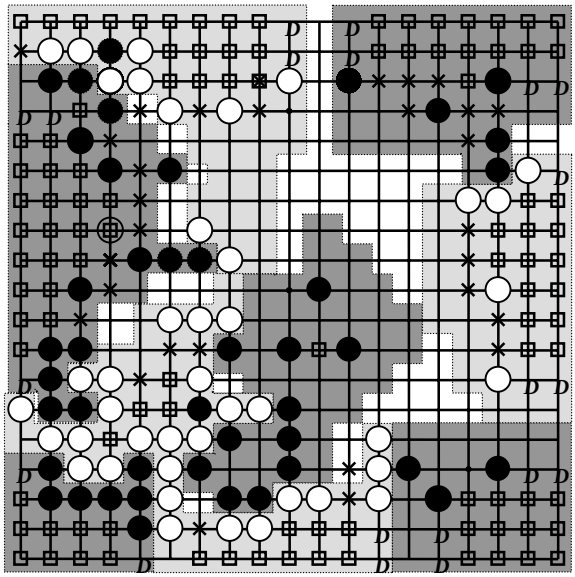

Fig. 16. The 7th Meijin title match (2).

Now consider the result of figure 16 computed by our system. The definite territories of White are 47 points and the definite territories of Black are 59 points. The points, which are under influence by White, are 44 points and under influence by Black are ahead. Cho says that Black is even closer to victory since Black has gradually whittled away at White's lower territory. This result is similar to the outputs of the system.

6. Future work

Pattern match system and string capture system are very important to our system. For the first subsystem, it needs a lot of human Go expert knowledge. For the other subsystem, programming technique is important. To improve the two subsystems will help the main system work more correctly. Identifying the degree of an unstable group is also a challenge. It is an important part of our system, too. We will discuss this problem in the future.

References

[Allis *et al.*, 1991] L.V. Allis, Van Den Herik, and H.J. Herschberg. *Heuristic Programming in Artificial Intelligence 2*, Ellis Horwood 1991.

[Berliner, 1974] H.J. Berliner. Chess as Problem Solving: the Development of a Tactics Analyzer. Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, 1974.

[Chen, 1989] Group identification in computer Go. In D. N. L. Levy and D. F. Beal, editors, Heuristic Programming in Artificial Intelligence: the First Computer Olympiad, pages 195-210. Ellis Horwood, Chichester.

[Chen, 1990] Ken Chen. The move decision process of Go intellect. *Computer Go*, No.14, pages 9--17, 1990.

[Cho, 1989] Cho Chikun. *Positional judgment high-speed game analysis*. Ishi Press, Inc. 1989.

[Hsu *et al.*, 1994] S.C. Hsu, J.C. Yan, and H. Chang. Design and implementation of a computer Go program Archimage 1.1. *Journal of Information Science and Engineering* 10, pages 239--258, 1994.

[Hsu and Liu, 1991] S.C. Hsu and D.Y. Liu. The design and construction of the computer Go program Dragon 2. *Computer Go*, No. 16, pages 3--14, 1991.

[Hwang and Hsu, 1994] Y.J. Hwang and S.C. Hsu. Design and implementation of a position judgment system for computer Go programs. *Bulletin of the College of Engineering, N.T.U.*, No. 62, pages 21--33, Oct. 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. *Game programming workshop in Japan*, pages 133--139, 1996.

[Lorentz, 1995] Richard J. Lorentz. Pattern matching in a Go Playing Program. *Game programming workshop in Japan*, pages 167--174, 1995.

[Müller, 1995] Martin Müller. *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory*. Ph.D. Dissertation, Swiss Federal Institute of Technology Zurich, 1995.

[Newell *et al.*, 1958] A. Newell A., J.C. Shaw, and H.A. Simon. Chess playing programs and the problem of complexity. *IBM Journal of Research and Development*, Vol. 4, No. 2. Pages 320--335, 1958.

[Reitman and Wilcox, 1978] Walter Reitman and Bruce Wilcox. Pattern recognition and pattern-directed inference in a program for playing Go. *Pattern-Directed Inference Systems*, pages 503--523, 1978.

[Ryder 1971] Heuristic Analysis of Large Trees as generated in the Game of Go. PhD thesis, Department of Computer Science, Stanford University., August.

[Sanechika 1991] The Specifications of "Go Generation" ICOT TR-720.

[Zobrist 1969] A model of visual organization for the game Go. In Proceedings of the Spring Joint Computer Conference, volume 34, pages 103 - 112.