TR-545

"Go Generation" A Go Playing System

by

N. Sanechika (AIR)

April, 1990

"Go Generation" A Go Playing System

N. Sanechika[*]
Electrotechnical Laboratory of the Agency of Industrial Science and Technology

H. Oki
Future Technology Laboratories Inc.

S. Yoshikawa[†] T. Yoshioka S. Uchida
Institute for New Generation Computer Technology

November 10, 1988

Abstract

This report is an introduction to the research and development of the Go playing system GOG (Go Generation) in the Fifth Generation Computer Systems Project. The purpose of this research is to pose basic problems of search problem solving, to resolve ambiguities, to transact exceptions, to develop cooperative problem solving in the AI field, and to find new approaches to conquering difficulties through the development of the Go program. GOG has been developed as part of the intermediate stage of the Fifth Generation Computer Systems Project. A working group, the AI scholars' organization, has been established as an adviser, and this research has been conducted jointly with it and the Electrotechnical Laboratory of the Agency of Industrial Science and Technology of MITI . We have been trying to simulate the human player with some themes of AI. In February 1987, a prototype that could finish the game was developed. Since then, we have been modifying the system and have transferred it from PSI-I to PSI-II. Now the system runs much faster, fast enough to play Go.

1 Introduction

Programming Go includes some basic subject which are the core of AI. It is possible to rate how well the new programming languages and computer systems can solve AI problems by researching and developing Go programs. Also, simulating human thinking on the computer may lead to a new approach to solving AI problems. Compared with Chess, Go has a vast search space, so the search must use the knowledge database efficiently rather than use simple search oriented methods. Consequently, we consider that programming Go provides a good case study which requires unconventional ideas and innovative AI technology. To pose these basic AI problems is the main reason why we selected Go programming as part of the Fifth Generation Computer Systems Project.

The following subjects are the main themes of research into Go programs.

1. Search: Extending the limit of the search's paradigm

[*]Currently at AI Research Institute Ltd.

[†]Currently at NTT Data Communications Systems Co.

2. Ambiguities: Managing the concept of ambiguities and introducing consideration of it
3. Exceptions: Dealing with new or unusual situations
4. Cooperation: Systematic integration of a cooperative system
5. Learning: Experiencing new things and extrapolating rules

We have adapted these themes, except learning, to GOG and explain them in more detail below.

(1) Search

Search is known as almighty method. In AI programming, search is one of the basic techniques. The mechanism of searching is to enumerate all candidates then find satisfied answers. This method is called "generate and test" and sometimes causes a combinational explosion. To avoid this problem, candidates must be well selected before the test. It seems a bit like going round in circles, but selecting the candidates at many steps makes solving overflow easier.

(2) Ambiguities

The concept of ambiguities and consideration of it can be defined as the main theme of fuzzy theory. Ambiguities come from the nature of human being. Ultimately, fuzzy theory is subjected to qualitative and symbolic processing rather than quantitative and numeric processing. Usually, scientific processing is determined as analytic processing, so painstaking analysis leads to the final goal. However, some human concepts become meaningless under such analysis, or cannot be found to exist. Also, people's limited time and memory mean that they often make a decision without complete information. Some abstractions of Go, such as Atsumi, and concepts based on them do not exist in nature.

(3) Exceptions

When a new situation occurs, human beings use their special body of knowledge and try to fix the situation. to simulate this mechanism, it is necessary to distinguish a regular situation and an irregular situation. An irregular situation can be either advantageous or disadvantageous. If it is disadvantageous, the first thing to do is find causes. The next thing to do is to prepare a couple of problem solvings for each cause. A standard of judgement in an irregular situation is different from one in a regular situation. Generally, a process in an irregular situation is under conditions that are more lax than those in a regular situation. For example, to leave a room, we normally use the door. However, if the door is jammed, we could try to get out through the window. As ways of getting out of the room, using the door and the window have the same result. Generally, in the irregular situation, the process has low efficiency and accuracy, so the process should be done according to the regular situation. Namely, the simulation should be designed on the assumption of many conditions. Experienced irregular events become known, and in that sense regular, so this theme is closely related to the learning.

(4) Cooperation

When human beings face a problem without a standard problem-solving theory for it, they closely examine all primary factors creating this problem, then try to make clear the relationships between the factors. Like this, human beings can organize parallel phenomena systematically. In Go, by judging from their opponent's last move and possibilities of their own next move, human players decide their next move. At this moment, they are organizing parallel phenomena systematically. For example, there are targets A and B. If we try to capture A and just work on it, we cannot capture it. Also If we try to capture B and just work on it, we cannot capture it. However, sometimes we have a chance to capture A while working on B. This example mentions the truth of cooperative problem solving.

We tried to simulate a human player, bearing in mind the above themes. Our GOG is still under development, so the following is an interim report. Our research has been conducted since 1985. In that year, development tools for GOG were prepared. In 1986, the concept of GOG was founded, and a prototype of GOG was created based on that. In 1987 and since, we have worked on the elaboration of position recognition methods, the expansion of search and knowledge, and the implementation of the cooperative problem solving system. The GOG program is about 40,000 lines long and is written in Extended Self-contained Prolog (ESP) running on the PSI (Personal Sequential Inference Machine) which has been developed at the Institute for New Generation Computer Technology (ICOT).

2 Modeling A Human Go Player

Generally, the game tree search approach using the Alpha-Beta pruning method has been widely adopted in developing two-person games with perfect information like Chess and Go. Research has concentrated on the game of Chess, so the present level of Chess programs is quite high. However, it has proved difficult to apply the search-oriented method for Chess to the game of Go, because the search space of Go is far larger than that of Chess. So programming Go requires an entirely new approach.

GOG is such an approach. It will simulate as accurately as possible the way the human player decides moves.

The way a human Go player plays is roughly divided into recognition of the positions, search, planning, and action.

(1) Recognition of Positions (Function of Visual Information Management)

In general, when someone plays Go, the function of visual information management is very important. There is a remarkable tendency for recognition of the positions to depend on it. The human player doesn't recognize the stones as being at coordinates so much as as members of a group that has tactical and strategical significance. This recognition evolves by experience or by learning from someone. The significant figures of stone groups and the player's learning process are stratified as the player's knowledge of Go.

In GOG, we use several of these groups of stones: point, string, group, and family. Also linkage, which is recognized as a connecting line.

Let's look at the process of forming these concepts. To play Go, the first thing to understand is point. Beginners soon understand about capturing stones and escaping from the enemy's attack. So the concept of string, a group of stones in line, is invented.

After a little experience, the player notices that strings that have escaped will often be captured immediately afterwords. This leads to the concept of group which may survive as a set of adjacent strings. Following this concept or in parallel with it, the concept of family is invented for a sphere of influence or a territory. Go is a game of territorial warfare, the concept of family is easily understood from the beginning. But players don't usually see how important it is until they have had a lot of experience because the concept of family is firmly related with typical Go strategies (such as KARAMI, and MOTARE) rather than territories. The concept of linkage consists of empirical knowledge. For example, if two stones are placed close together, they can be connected easily from the enemy's point of view. Also linkage is recognized as the border line, so enemy stones that invade the territory are cut off from communication with the outside by linkage.

All the above concepts will have meaning only together rather than in isolation. The human player recognizes these group shapes at a glance.

(2) Search (Function of Fact Confirmation)

Sometimes an unstable place appears in a small area of the board such as a linkage nearly cut, a group nearly dead, or two struggling groups. When settlement of such a place will change the balance of power in the game, reading moves is necessary. GOG reads moves for the first and the second move. GOG decides whether the two moves together will succeed or fail, all GOG has to do is revise data structures. If the first move will succeed and the second move will fail, the game will turn into busy and the first move will be rate highly. It is necessary to recognize the positions and look for related candidates to read a move (such as "capture" or "cut") with a specified intention in a local area. Methods other than reading moves, such as specified pattern matching, cannot cover various kinds of changes. But reading moves costs a lot of memory and takes a particulary long time. Nevertheless we adapt that method to GOG by limiting it for local use and by not using it until the problem has been well considered.

(3) Planning (Function of Inference based on Ambiguous Knowledge)

In Go, there is no method to find the best move during planning because the algorithm which always leads to a win hasn't been found yet. The player chooses a proper plans (for the player) from some possible plans and makes a decision based on foresight and taste. The player chooses the plan most likely to materialize and is expected that it changes the balance of power in the game. But still there are too many candidate plans, considering only these conditions. Actually, not all these candidates are examined. A player seems to adopt a plan which is based on that the player remembers working in a similar situation. Based on this idea, GOG introduces the concept of CASE which is newly created by combining Go's typical situations and their related tactics.

During planning, choosing between a plan which could gain a big advantage but for which a series of actions has not yet been decided and one which offers less advantage but has an exact plan of action is a gamble. Players generally think that the necessary actions will become obvious after planning but this is not always the case. Making a plan without any idea that materialize the plan is wasting time. Also deciding the details of the actions during making a plan is laborious and takes a lot of time. Many players takes the middle road by planning and devising a tentative course of action simultaneously. The accuracy of this course of action corresponds to the number of candidates of the plan; the more candidates, the lower the accuracy. When there are many candidates, the process of deciding the actions takes several steps. At the end, actions that are accurate enough are offered when planning is done. This is an expression of the principle of iterative deepening recognition. In GOG, a set of actions (a candidate) corresponds to a provisional actions in the first layer of iterative deepening.

Even if players decide a plan, the action for its realization is still not decided by a single option. The plan always keeps open the possibility of optimization. When players are deciding on a course of action, sometimes they find a good way, but then find an even better one. In such a case, we don't stick with the first course of action.

(4) Action Structure (Function of Systematic Complex Phenomenon Integration)

Human players use methods of structuring a course of action that is different from the game tree search in two-person games with perfect information. Some characteristics of that method are listed below.

- Object/Target Directional Read

  The maximizing static evaluation function on a standard search is general and abstract, but human beings aim at a definite object and look for it.

- Integration of Method Decision and Action Structure

  Gradually, as explained in (3) above, the search narrows and a course of action is pinpointed.

- Reading Different Moves Simultaneously and Merging

  Assembling prospects which haven't been approved yet or a few prospects, one of which will be selected.

- Object/Target Alternations Part Way through Process

  Choosing a higher object or target than the present one, when the human player finds one while reading moves.

3 GOG Program: Modeling A Human Player

To simulate functions human player, GOG use the module structures illustrated in Fig.1.

3.1 Recognition of Position

3.1.1 Revision of Data Structures

Abstract subject on the board constitute data structures. Revising data structures in incremental revision means that the version is only implemented on those data structures that were changed by the last move. This approach saves time, but involves complicated procedures. So when it is not necessary to consider time, we use overall revision instead. Data structures of GOG are associated with individual points, strings, groups, families and linkages. The various data structures are represented in Fig.2.

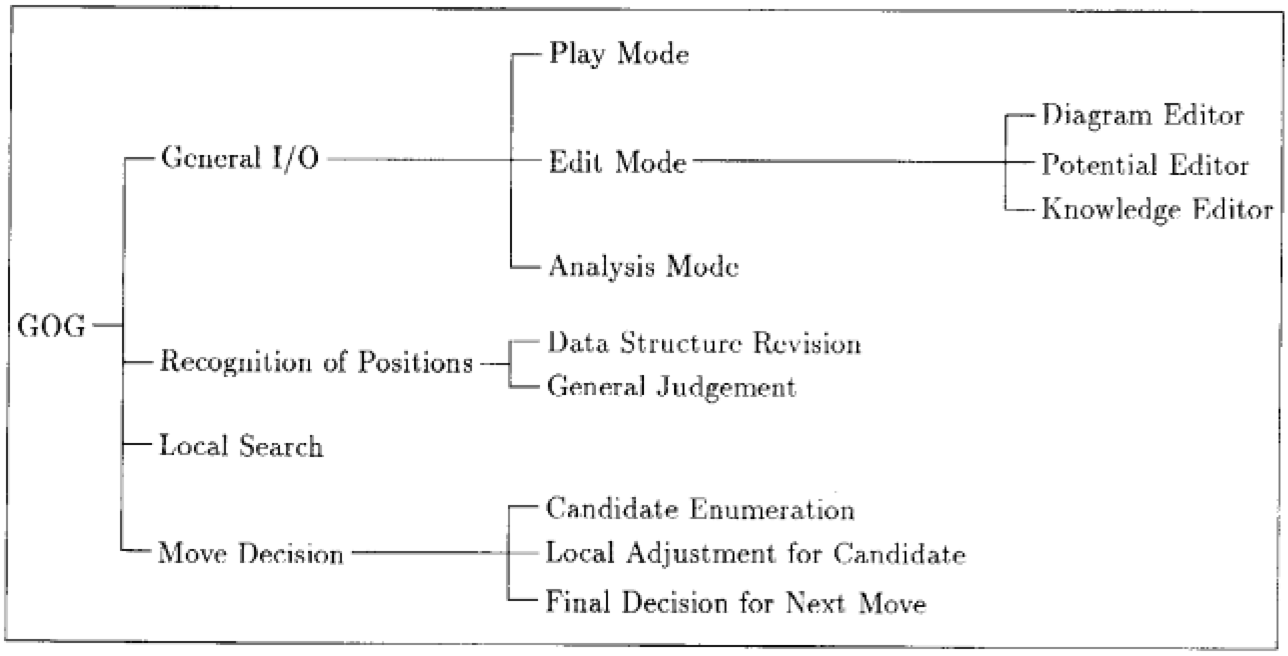


Figure 1: GOG Module Structures

Each object has some attributes. Here, we'll explain one of them, rate of siege, which is a measure of the possibility of besiegement. It is a rather ambiguous concept, but one of most important factors for judging the vitality of groups.

Fig.3 shows black stones becoming surrounded by white. Placing one or two more white at the upper side surrounds black completely. We have developed an approach like Fig.4 so that the situation can be recognized by a computer.

In Fig.4, the search is spreading from each stone in the group to the group's DAME. The number of apertures are defined as group's DAME and 4th group's DAME are defined as the rate of siege. When the search reaches an enemy stone or blocked point, which is a point on enemy linkage, it will be stopped. The search starts from the group which is nearly surrounded and rates the aperture of siege.

In Fig.4 4th group's DAME is 6, so the rate of siege is 6. The lower the rate of siege, the stronger the besiegement. Zero is a perfect siege. Every 4th group's DAMEs' positions will be used by the search for the possibility of escape from siege.

The following is the definition of the rate of siege written in ESP.

```
rate_of_siege(Go,E):-group_dame(4,Go,Gds),size_of_list(Gds,E);

group_dame(N,Go,Gds):-N>=1,!,group_dame(N-1,Go,Gds1),
        group_dame_dame(Gds1,N,Gds);
group_dame(0,Go,Gds):-group_stone(Go,Gds);

group_dame_dame([],_,[]):-!;
```

Object	Main rule	Function	Main attribute
Point	Board configuration	Stone's location	Color,contact point, potential,candidates
String	A unit which is alive or dead simultaneously	Unit of capture	Number of stones, DAME points, assigns life and death
Linkage	Supposed line between two points of the same color or between a stone and the board edge	Border of territory unit of connection	Type of linkage, type of connection
Group	A unit of the same color whose stones are strongly conected	Unit of life and death	Number of stones, territory size,vitality, rate of siege
Family	A unit of the same color whose potential has more than specified value	Unit of territory	Number of stones, length,territory size, strength,importance

Figure 2: Position Data Structures and Attributes

```
group_dame_dame([Gd|Gds],N,Gdsn):-group_dame_dame(Gds,N,Gdsn1),
        adjacent(Gd,AGds),spreadable(AGds,N,Gdsn2),
        append(Gdsn2,Gdsn1,Gdsn);

spreadable([],_,[]):-!;
spreadable([AGd|AGds],N,[AGd|Gds]):-spread_condition(AGd,N),!,
        dame_flags(AGd,N),spreadable(AGds,N,Gds);
spreadable([AGd|AGds],N,Gds):-spreadable(AGds,N,Gds);

spread_condition(Gd,N):-dame_flags(Gd,N0),N0<N,color(Gd,empty),
        not(blocked_point(Gd));
```

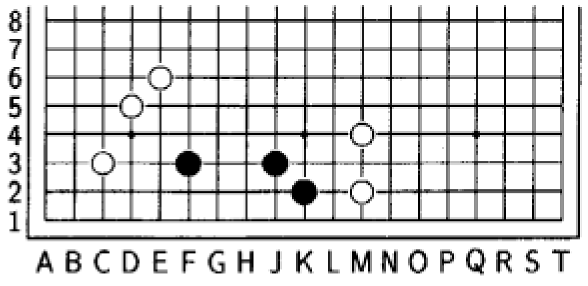
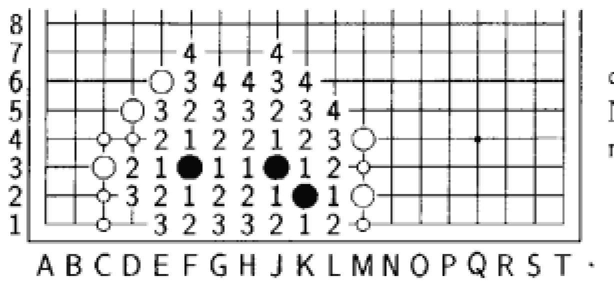


Figure 3: Image of Enclosing



o	: blocked point
N	: Nth group's DAME
rate of siege	: number of 4th group's DAME

Figure 4: Example - Calculation for Rate of Siege

Sometimes revision of data structures cannot be finished by just one procedure. More revision is often necessary to fix inconsistency between individual data structures and to achieve deep recognition. Take a look at "captured string adjustment" and "dead group adjustment".

Object	Condition	Main function
Captured string-adjustment	When the other color's possible captured string is made	Start capture search considering struggle
Dead group-adjustment	When dead string is made	Merging adjacent enemy's group and resetting all potential

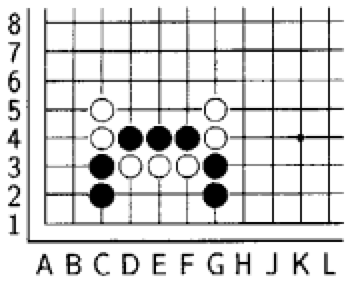
Figure 5: Data Structure Adjustment

Captured string adjustment improves the recognition of situations in which strings can be captured. Dead group adjustment improves data structures using the effects of dead groups. For example, in Fig.6a, the three white and the three black stones in the middle of the board are recognized as dead stones without captured string adjustment. Because the local search does not use the results of fights between these six stones, this absurd result occurs. If there is any enemy stone, which has the same extent of weakness as the target stone of the local search has, near by, the search should consider moves that could capture the enemy stone. Searching from the wide point of view at the beginning saves trouble but the cost of the process is going to be much more. So the best way is do the phased search, if it is necessary.

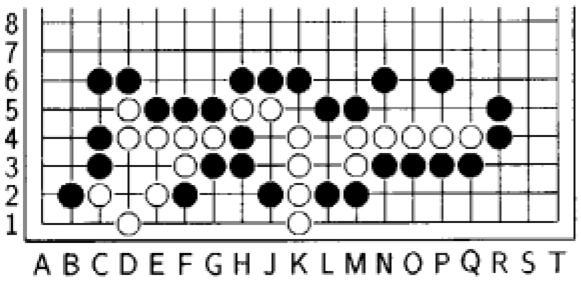
In Fig.6b, the group of white stones on the right appear vulnerable, but they are safe because the five black stones in the middle are dead by being surrounded. Actually, all the white stones are making one big group. The problem is the condition of the stone, dead or alive, is judged by using the concept of group in the data structure. Solving this, first of all, we define the basic small groups of stones as temporary groups and record all attributes of them, then we redefine a dead group and its enemy neighbor, if such exist, as one new group. At the same time, the potential value and attribute of the dead group are reset.

3.1.2 Judgment of the General Situation

Judgement of the general situation is a result of recognition of position. It contains the progress status (opening, middle, and end), the game situation (tactics of defense and offense, and so on),



(a) Captured String Adjustment



(b) Dead Group Adjustment

Figure 6: Data Structure Adjustments

and the game status (winning, even, and losing). It won't be used for selecting candidates directly, nonetheless, it helps to avoid making an unnecessary search tree.

The game status is mainly used for deciding the degree of offense and defense. GOG has adopted the Chinese rules which define a player's territory as consisting of all the points he has occupied or surrounded. The Japanese rules define territory to be not occupied or surrounded points but vacant points surrounded by independently live groups.

Counting firmly living stones and the territories surrounded by them, and counting dead stones as the opponent's territory are both easy. The problem is counting possible struggle spheres. The basic idea of solving it is to calculate the rate of the potential of all positions on the game board by using custom formulas created for stones that exert influence. The potential defines the strength of the stone's occupancy at each position. A position which has large potential is likely to become occupied. Also Fig.8 shows the formula which exchanges rate of potential to "size of territories".

Fig.9 is the result of performing this process on Fig.7. A position which is empty but recognized as someone's territory is marked with a small white circle as white's territory or a small black circle as black's. The total of stones and marked places becomes each team's territory.

3.2 Local Search

3.2.1 Function of Local Search and its Materialization

Local search is necessary at many stages, but actually these searches can be organized as one program from the programmer's point of view. If the position, the local search objects (targets) and the player have been specified, the conditions for the search can be specified. So all the program needs is a tool that has these parameters. However, each object has a different heuristic method for creating candidates and arriving at final decisions for them. So customizing search modules for each object could give better performance. Fig.10 shows each objects' search modules. The type of target are specified by each purpose of the search, such as the target of the capture is the string.

The algorithm for the local search is adapted from the Alpha-Beta pruning method. The search continues until there is no alternative, so the result must be success or failure. Generally, using a search routine requires more run-time. If the routine is used a lot, it takes up a lot of CPU time and memory. So the routine must have well established conditions to start it (Fig.10). These conditions are quite useful for checking the necessity of starting the search and for specifying the

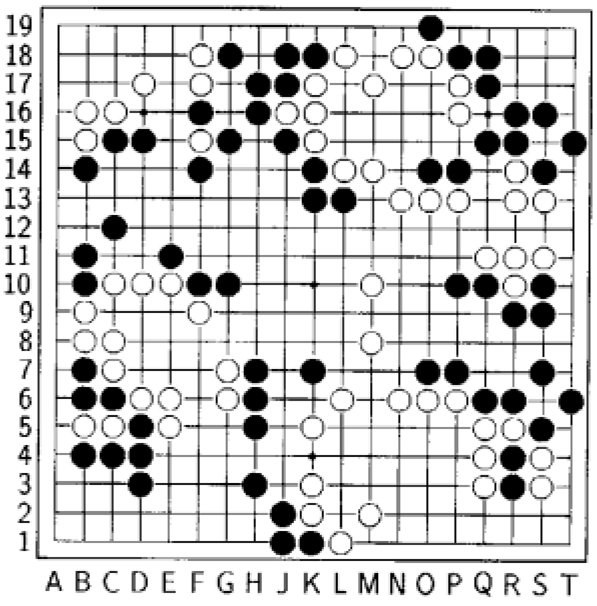


Figure 7: Diagram for Situation Judgement

search area. The procedure of the check must be as simple as possible while satisfying the condition. The following explains the Alpha-Beta pruning method after Negamax representation in ESP.

```
negamax(P,_,_,V,_):-terminal_position(P,V),!;
negamax(P,A,B,V,M):-candidates(P,Ms),
        best_candidate(Ms,P,B,A,_,V,M);

best_candidate([],_,_,V,M,V,M):-!;
best_candidate([M|Ms],P,B,V1,M1,V2,M2):-move(M,P,NewP),
        negamax(NewP,-B,-V1,MV3,_),V3=-MV3,
        (M3>=B,!,V2=V3,M2=M3;
         M3>=V1,!,best_candidate(Ms,P,B,V3,M,V2,M2);
                best_candidate(Ms,P,B,V1,M1,V2,M2));
```

In this algorithm, "generate" represents candidates and "test" appraises the positions that are arranged for each search object. Defining these also defines the search itself. Now we explain SHICHO, complete search and HOKAKU, heuristic search.

(1) SHICHO

SHICHO (ladders) is a sequence of moves that ends in disaster for one side or the other, depending on conditions several moves ahead. For example, in Fig.11a, black threatens to capture white by placing a stone on position 1.

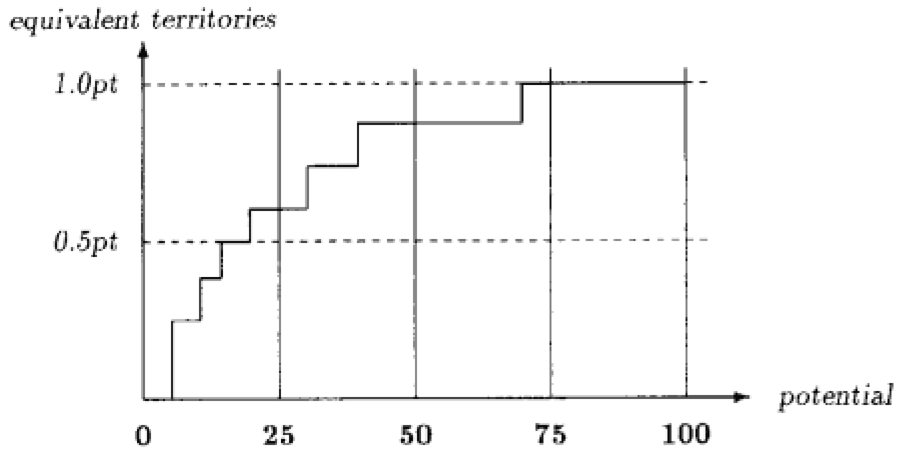


Figure 8: Equivalent Territories Function

When white attempts to keep an escape route open by placing a stone on 2, black places on 3, and so on. The white stones from a staircase that is eventually trapped against the edge of the board and the whole group of white stones is captured by black. Each of blacks' moves in this example are called ATARI because they force white along the ladder to disaster.

Fig.11b and 11c show OIOTOSHI and UTTEGAESHI (snapback) two more typical Go techniques. They also use a series of ATARI move, so they are special forms of SHICHO. Once SHICHO is recognized, the decision for the end of search and candidates is clear. The assessment for the end of search is 1 if SHICHO will succeed and -1 if it will fail. So if there are 2 DAME in offense or there is 1 DAME in defense, the search continues until these conditions become different.

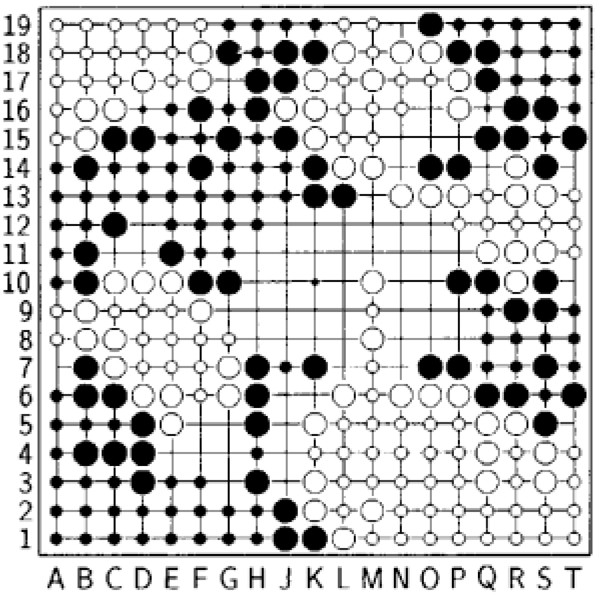
Fig.13 shows candidates for SHICHO.

In Fig.11a, like 1, 3, or 5, if white moves one of these, the white stones are going to have 4 or more DAME; this comes under a candidate (1) in offense of Fig. 13. If this move is possible, wherever black takes SHICHO won't be success. (1) is included in (2), so it is not quite necessary to define it, but defining (1) makes SHICHO procedure more efficient. This condition makes SHICHO go directly to it's conclusion.
Also because (2) is defined in defense, the defense side doesn't only escape from capture, but also tries to capture the opponent. So OIOTOSHI and UTTEGAESHI (snap back) are also easily done, just like SHICHO.

(2) HOKAKU (capture)

HOKAKU operates upon all problems except problems in which a target string will be enclosed completely (such as TSUMEGO and mutual struggling problems).

Fig.14a, 14b, and 14c are examples of HOKAKU, which has fewer limitations than SHICHO.



Situation judgement

	Black	White
Territories	149 points	146 points
KOMI		5.5 points
MOYO	8 points	6 points

White leads 0.5 points

Figure 9: Example for Situation Judgement

A selecting heuristic is necessary for improving the search efficiency. We particularly need a heuristic which speeds up the search or controls enumeration candidates.

Fig.15 is the heuristic condition for end of search. In the Figure, (1) says "If a target string has 4 DAMEs or more, stop the search." if we change the number of DAMEs to a much bigger number, we could search through a larger selection of candidates. Although we can search more candidates, the problems that are solved by this search do not increase as much as the search time does. (2) also has more limitations. If a target string has only 3 DAME but 3 or more stones in an enclosing string can be captured easily, the target string won't be captured. Giving this condition helps the efficiency of search a lot. However, we could still have an explosion of search. (4) stops the search and returns 0 as the assessment, if the search procedure reaches specified depth and takes specified steps.

Fig.16 shows the heuristics of selecting candidates. It has normal moves and compulsory moves. If the present position satisfies any condition of a compulsory move, the move becomes a candidate. If there is no compulsory move, all normal moves become candidates. In Fig.14a, the first move is selected from compulsory move (1). In Fig.14b and 14c, the correct move is one of the candidates selected from the normal moves (3) and (4). These problem will end as SHICHO after filling DAMEs up by normal move (1). In addition, the basic DAME point, which is assessment of normal move (1), is defined by 10 / (target string's DAME), and the increment is the increment of DAME after placing a stone.

Object	Target	Start condition	Result
SHICHO	String	2 or fewer DAMEs	Alive, dead
HOKAKU	String	3 or fewer DAMEs	Alive, dead, unknown
Connection	Linkage	Opponent comes to peep near linkage	Connect, cross, through
Life and death	Group	N or fewer empty points in perfect siege	Alive, KO, dead
Struggle	Both B/W groups	Opponent group satisfying start condition for life and death	Alive, KO, dead

Figure 10: Local Search Modules

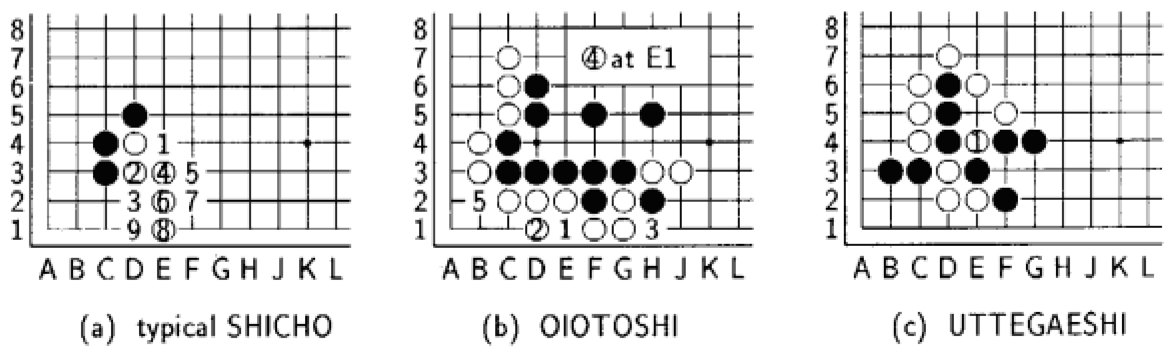


Figure 11: Examples for SHICHO

3.2.2 Difficulties of Local Search

With local search, improving its performance is important, but ways to use it are also important. The following difficulties occur with it.

[1] Selecting the Search Object

Selecting the search object is done by conditions in Fig.10. This is like the pre-search for the main search. When there is a possible search object, all factors of the object are defined by knowledge of the object.

[2] Need to Apply Result of Search

Results of the search are used in many ways, but there are two main applications. These are applied in the recognition of positions as a supplemental expedient and as the move decision. Fig.17 describes the usage of the local search at HOKAKU as an example. The arrows in the figure show each module calling its search module. the purposes are as follows.

(1) To define the string's attribute of life and death (it is going to be used by defining belonging attribute of string and group and enumerating candidates for capture

Terminal position satisfies one of the following, and returns its assessment	Assessment
(1)In offense, target string has 3 or more DAMEs	-1
(2)In offense, target string has a DAME	1
(3)In defense, target string has 2 or more DAMEs	1

Figure 12: Conditions for End of SHICHO

Candidates in offense	Candidates in defense
(1)If defense moves here, DAME will be 4 or more	(1)Target string's DAMEs
(2)Any move at target string's DAMEs except (1)	(2)Will capture adjacent enemy string

Figure 13: SHICHO Candidates

and escape).

(2) To search inconsistencies among neighbor strings, then start wider search, if any.

(3) To use for end-search or for creating a candidate.

(4) To estimate the strength of connection by the general judgement which uses results of HOKAKU search for crossed strings' life and death.

(5) To not place any stone, if a temporarily placed stone will be captured, unless you meant it to be wasted.

(6) To move to the position that has the highest assessment around there, if there is any candidate that has the intention of capture and the position has the same intention.

There are two more purposes.

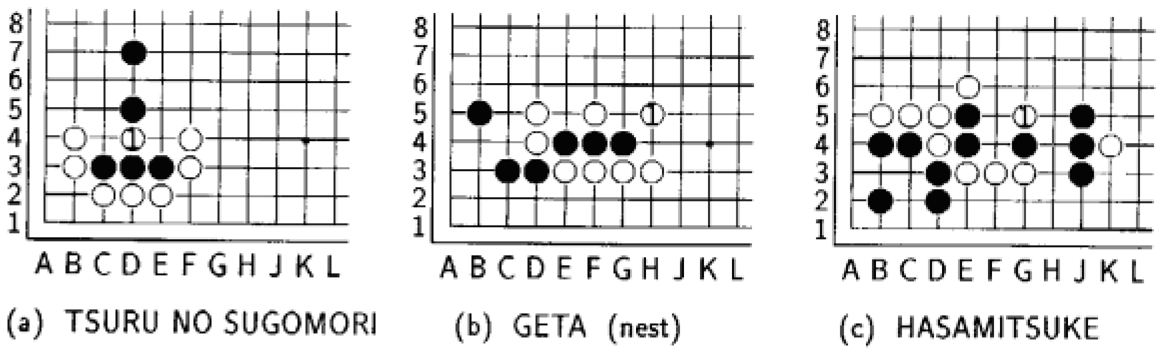


Figure 14: Examples for HOKAKU

Terminal position satisfies one of the following, then returns its assessment	Assessment
(1)In offense, target string has 4 or more DAMEs	-1
(2)In offense, target string has 3 DAMEs, 3 or more adjacent enemy strings have 2 or fewer DAMEs	-1
(3)In offense, target string can be captured by SHICHO	1
(4)If search's steps and depth exceed limits	0

Figure 15: Heuristics for End of HOKAKU

Offense's heuristic	Assessment
【Compulsory moves】	
(1)If defense moves here, DAME will be 5 or more	
(2)If target string has 3 DAMEs, move at enclosing string (size2 or more)'s ATARI	
【Normal moves】	
(1)Any move at target string's DAMEs	Basic DAME point x increment
(2)Move at enclosing string's ATARI	9
(3)Target string's KOSUMI (height 2 or more)	10
(4)A space skipped from target string (height 2 or more)	8
⋮	
Defense's heuristic	**Assessment**
⋮	

Figure 16: Example for Heuristics of HOKAKU Candidates

(7) To score own HOKAKU exercise

(8) To display the process of HOKAKU search

As above, a routine can be called by a different kind of routine or can call a different kind of routine.

[3] Managing Result of Search

Sometimes the result of search is used only one time, and sometimes it is used many times until the disposition of the position is changed. However, if conditions of search change after the game's progress, because re-recognition of the position is necessary the disposition becomes unreliable. If that happens, what kind of situation destroys the judgment once made? Examining this has a lot of difficulties. It seems that the stone's influence is exerted mainly on its neighbors, so all we have to do is to search the condition of positions that are related to objects near the last move that occurred. However, this will not usually succeed. A typical example is a move which prevents SHICHO, also some string's capture causes a SEKI break and a chain reaction spreading its influence

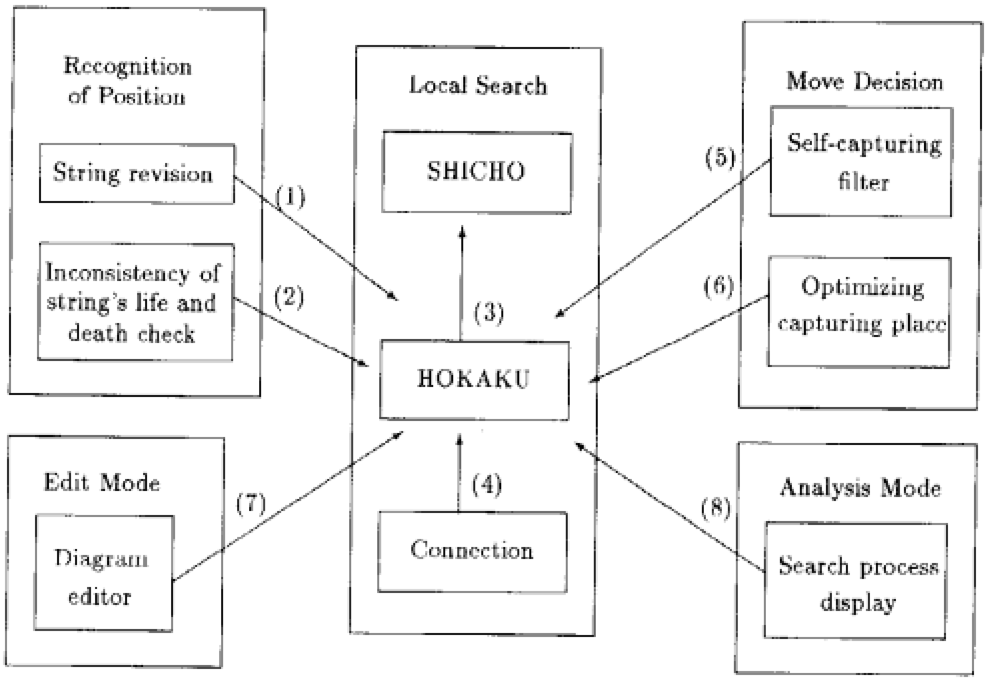


Figure 17: Flow of HOKAKU Search Process

over a wide range. Currently, GOG could expand its search as much as its knowledge covers.

3.3 Move Decision

The process of move decision is as follows. First, GOG implements position recognition and makes up a plan based on its results. Then it examines possibilities of materializing the plan and repercussion effects considering the full extent of its influence and side effects. Fig.18 describes a flow of GOG's decision making.

3.3.1 Enumerating of Candidates

We assume that human beings act based on their knowledge of experienced typical situations, when they are faced with problem solving. For GOG, we call the concept of these typical situations CASE. It means the point of decision making is selected by CASEs established by knowledge of Go. Decision of what kind of CASE is currently occurring is done by defining traits of each CASE. Also, when a CASE is detected, a candidate (from now on, we call it a CASE candidate) as a device for managing CASE and the method of ordinary assessment are defined. By doing this, the intention of move decision (specially opponent's) and assessment of move can be understood. Fig.19 shows CASE candidates. We'll talk about the enclose/escape move and the FUTOKORO (local territories) reduce/expand move.

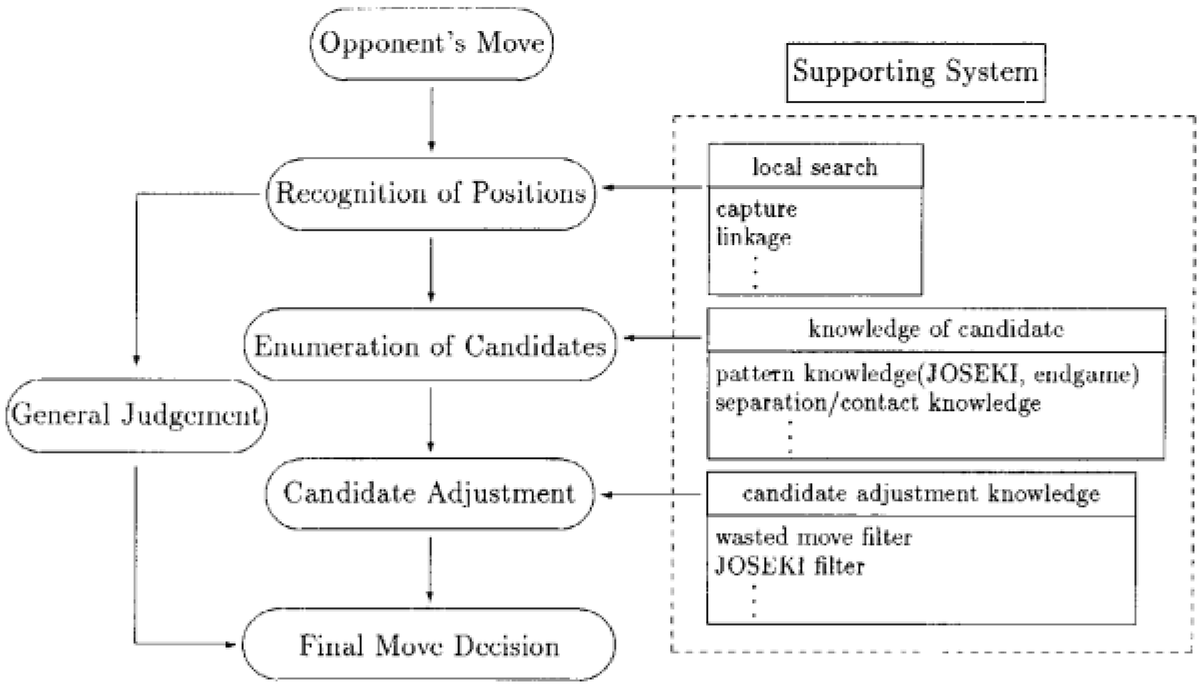


Figure 18: GOG's Module Structure with Flow of Decision Making

Like Fig.4, the enclose/escape move considers a group that has been nearly enclosed. From the figure, it seems that escaping through the middle of 4th group's DAME is the best way. However, placing a stone far from the group brings the opponent's cut move on the connection, while placing one too close to the group doesn't help at all. So, we looked at the Go saying "Skip a space (IKKEN), when you escape." We settled this as follows. First, GOG looks for a central coordination of the group's DAME, and it becomes the 1st approximate enclose/escape point. Then the nearest position in the group to it becomes a (enclose/escape) base point. An IKKEN point, KOSUMI point, or the nearest point of NOBI (a possible extension of the string) from the base point is defined as an enclose/escape point. In Fig.4, the 1st approximate enclose/escape point is J6, the base point is J3 and the enclose/escape point is recognized at J5. Their assessment is defined by the strength of group, its size and so on.

Next, we explain the FUTOKORO expand/reduce move. Examples for it are the ×s in Fig.20. These moves expand and reduce territories even in an ordinary situation, so using these moves takes advantage fairly. The importance of the moves increases when the life and death of a stone depends on them. H2 is not so important right now, but once black encloses the left white, it becomes very important. Like this, searching the pattern of moves can be done as the local search. However, its importance is decided from the general point of view. So the FUTOKORO expand/reduce move takes the following two steps. After a move has been made, GOG tries all patterns matching around the placed stone for the FUTOKORO expand/reduce move and adds new candidates to previous candidates or delete expired candidates from them. Then GOG examines any changes in assessments of FUTOKORO expand/reduce moves all over the game board.

The knowledge of the FUTOKORO expand/reduce move consists of two rules (candidates selection, assessment calculation) and the instance data in Fig.21. The candidates selection rule consists

Object	Characteristic	Contents
JOSEKI	Corner pattern	JOSEKI move (239 moves)
Edge position	Edge pattern	Edge move (HIRAKI, (TSUME, WARIUCHI, etc)
DAME (liberty)	Contact pattern	Mutual struggle move (HANE, NOBI, OSHI, etc)
Invasion	Corner pattern	Invade/protect move at (3,3)
Endgame	Mainly edgepattern	Endgame move
FUTOKORO expand/reduce	Mainly edge pattern, weak group	Moves for weak stone's FUTOKORO
MOYO tangency	MOYO pattern neighboring families	Move at MOYO's adjacency (RYO-GEIMA, etc)
Capture/escape	Strings (3 DAMEs or fewer)	Capture/escape move
Connection/cut	Being peeped linkages	Connection/cut move for peeping
Enclose/escape	Slightly enclosed weak friend group	Enclose/escape move for weak stone
Separation/connection	Adjacent weak friend group	Separation/connection move for stone
MOYO expand/reduce	Being peeped families	Enclose/invade move at MOYO

Figure 19: CASE Candidates

of local pattern rules such as the ×s in Fig.20. The assessment calculation rule is for calculating assessments and also will record all information of the recognition of position, such as conditioned reflexes.

3.3.2 Local Adjustment for candidates

The local adjustment for candidates rearranges disharmonies between the different CASEs. It has 5 different functions (see Fig.22). We'll look at the dead group filter and the capture position optimization.

First, the dead group filter. In Fig.23a, the white stones on the left are recognized as not being a dead group yet, so two KOSUMI moves are indicated as candidates for white. However, the white stones on the right are recognized as being a dead group, so no candidates are offered even though the pattern of stones is the same as on the left. This move signifies YOSE (endgame) and intensifies FUTOKORO to protect the weak white group, so the assessment of this move is the total of what the assessment of YOSE and the assessment of intensification of FUTOKORO.

Now, in Fig.23b, black to play, black has to protect D3 from white capture. At first, the capture routine suggested placing at E3. But D2 signals the need to intensify the FUTOKORO around E3, and the capture routines confirms that black can also escape from white capture by placing a black stone at D2. Accordingly, D2 has higher assessment as an escape move than E3 does.

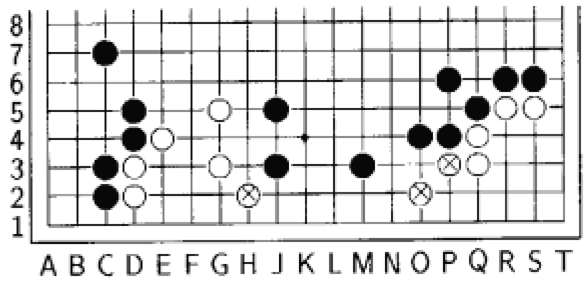


Figure 20: Example for Knowledge of FUTOKORO expand/reduce Candidate

3.3.3 Final Decision for Next Move

Candidates which were suggested by the different CASEs and had local adjustment at candidate adjustment have been narrowed down to one single next move. These processes are shown by the score table in Fig.24.

Each assessment of a CASE's candidate is recorded in the score table. If a place might have many CASEs, the assessment of the place will be the total of assessments of all CASEs. Fig.24 shows assessments of white's candidates in the table.

For example, a white stone placed at Q15 simultaneously helps to enclose black's group at R16 and expand white's MOYO. The result is that L14 has the highest assessment. However, this is still basics; sometimes the method of decision needs adjustment depending on general judgment or general knowledge of tactics. Generally, when you are winning, you had better control fights quietly as much as possible. When you are losing, you had better spread fights all over the board. Also selections of moves are limited by the game situation.

For example, when a Go player has a on thick outer shell and is winning, he may choose to protect the middle of his MOYO from enemy invasion or to enclose his MOYO by letting his enemy enclose the enemy's own MOYO. At this moment, if he has weak stones and they won't be changed at any situation even if they are taken by the enemy, and also if his escaping move will be toward his large MOYO, the priority of moves which rescue the stones goes down. Or if he is weakened having his weak stones taken, he stays away from his MOYO and tries not to waste any moves. In particular, when the enemy invades his largest MOYO, he keeps attacking the invader until the situation is settled.

On the other hand, if a player is in a weak position, he tries to invade his enemy's MOYO either by directly placing a stone in it or by breaking through his enemy's wall. Also he does not waste energy on small and weak enemy stones which don't influence winning and losing.

Comprehending general judgments like the above into all the situations is very difficult, but it increases GOG's strength as a Go player even in typical situations.

4 Development Tools

The development tools are not directly related with the main system of GOG, but shortening the term of development of GOG and improving the quality of GOG are depend on the quality of the tools. So development tools are very important. Programming of Go games must be improved

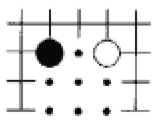
Candidates selection rule	Assessment calculation rule
standard_pattern : - friend(-1,1), enemy([(-1,0),(-1,-1),(0,1),(0,0),(0,-1),(1,0),(1,1)]), board_edge(0,-1), candidate_select((0,0),1); 	1 : - base point(BOTH,5), (weekstone(-1,1) then fighting point(BOTH,10)); 2 : - . . . . ,
	candidate instance data
standard_pattern2 : - . . . . ; . . . . ;	base coordination : H2 rotation of axes : 0 candidate coordination : H2 assessment calculation : 1

Figure 21: Knowledge of FUTOKORO expand/reduce

through trial and error and the size of program tends to be getting larger, so it is not an exaggeration to say that the success of this project depends on the quality of the development tools. General I/O is designed that GOG users can switch around between Play Mode and Development Mode (Edit Mode and Analysis Mode). It is also quite important for efficiency of debugging. Now we'll talk about things that are recognized as important in experiment.

4.1 Analysis Mode

This mode takes charge of debugging. Analysis Mode and Play Mode should be on-line connected and able to be switched at the user's discretion for directly examining the problems which occur during play. Analysis mode requires the peep function which can investigate all objects and their attributes in data structures. It also requires the trace function for the sub-module in case it has any problems. Fig.25 shows the module structures of analysis mode.

The following are explanations of each module's functions.

(1) Display of Data Structures

Displays a distribution of each data structure and its attributes which are represented by recognition of the positions.

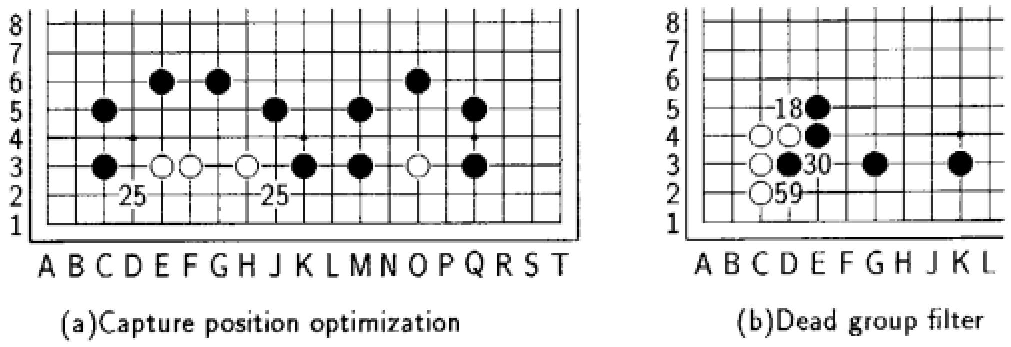
(2) Display of Candidate Move

Displays contents of the score table which are represented by candidate enumeration and the local adjustment for candidates. Also displays the equivalent of the assessment of each candidate by equivalent territories function.

(3) Display of the Situation

Object	Main function
Wasted move filter	Remove a candidate whose signification by its location is impossible to actualize
JOSEKI filter	Remove candidates except JOSEKI move, when GOG is under JOSEKI's control
Dead group filter	When a candidate makes a dead group and the move doesn't have any meaning for the dead group, remove the candidate
Self capturing filter	Remove a candidate which captures itself
Capture position-optimization	Select the best move when many candidates capture the same target

Figure 22: HOKAKU Candidate Adjustment



8
7
6
5
4
3
2
1
25
25
A B C D E F G H J K L M N O P Q R S T
(a)Capture position optimization
18
30
59
A B C D E F G H J K L
(b)Dead group filter

Figure 23: Examples for Candidate Adjustment

Displays the situation which is represented by the situation judgment.

(4) Display of Search Process

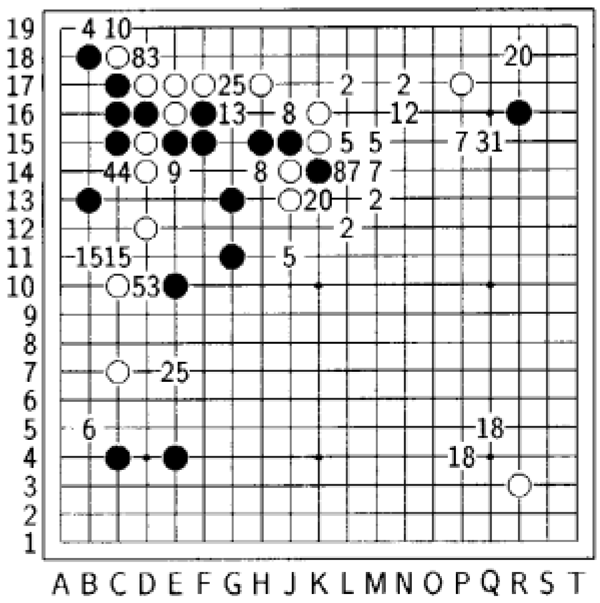
Using each different search module, displays these search processes and their results. Also it is possible to display the order of their priority.

(5) Tuning of Parameter

The knowledge of recognition of the positions or the candidate enumeration contains many parameters. Proper adjustment of each parameter is very important for accurate knowledge.

4.2 Edit Mode

Edit mode edits Go problems for Analysis mode, improves the knowledge of JOSEKI and improves common moves.



B5	endgame	K13	extension
B11	block	L12	separation
B19	extension	L14	capture, connection
C11	connection	L15	extension
C14	extension, block	L17	connection
C19	block, extension	M13	separation
D10	connection, escape	M14	MOYO, separation
D18	connection, escape	M15	MOYO
E7	escape	N16	HIRAKI
E14	extension	N17	connection
G16	extension	P4	JOSEKI
G17	invasion, extension	P15	MOYO
H14	extension	Q5	JOSEKI
J11	MOYO	Q15	enclose, MOYO
J16	extension	R18	endgame

Figure 24: Score Table

Fig.26 shows the module structure of Edit mode.
The following are explanation of each module.

(1) Diagram Editor

Creates the position as a problem and Preserves/Regenerates it for debugging GOG. The problem for search (HOKAKU, TSUMEGO and so on), move decision (for the next move and so on), and recognition of the positions (judgement of the stone's strength and so on) can be registered in self-judgement. Also during self-judgement, we can examine the changes of answers after changing GOG's parameters.

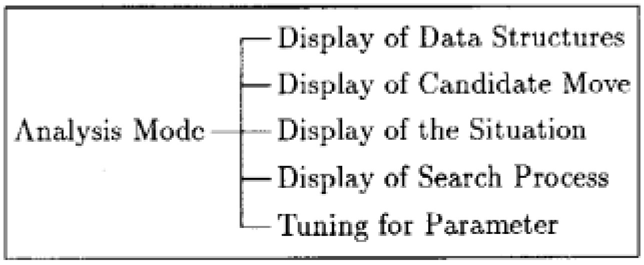

Analysis Mode
Display of Data Structures
Display of Candidate Move
Display of the Situation
Display of Search Process
Tuning for Parameter

Figure 25: Module Structures of Analysis Mode

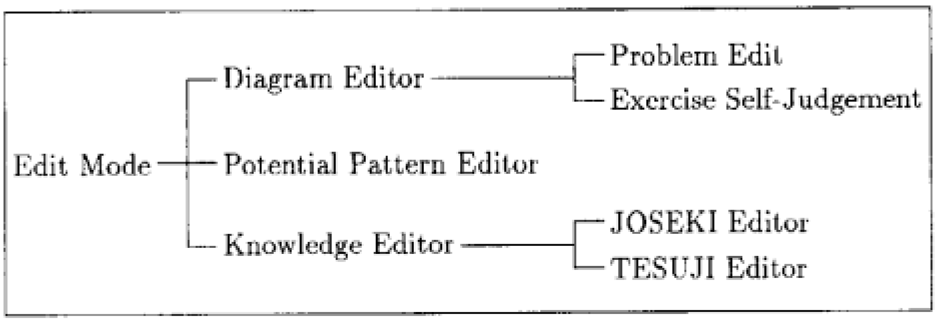


Figure 26: Module Structures of Edit Mode

(2) Potential Pattern Editor

The potential expresses the influence a stone exerts on its neighbor. However, this influence is affected by the corners and edges of the board, so it cannot be decided impartially. GOG has prepared some potential patterns corresponding to a stone's location on the board. We set up that distribution so that each of those patterns' potentials could be modified by an editor.

(3) Knowledge Editor

This editor is established by the JOSEKI editor and TESUJI editor. JOSEKI data is prepared as a large tree structure. Each branching point of the tree has a condition of JOSEKI (ATSUMI, SHICHO-oriented and so on).

Data of TESUJI is organized by filed data for each similar move. Each single data contains applicable conditions (timing, location, and stone's arrangement) and process of matching and its knowledge of assessment.

5 Assessment of Present GOG and Its Problem

We explained 5 main themes of AI in the first paragraph. Now we explain what problems we have solved or not yet done on GOG from the point of view of these themes(except learning).

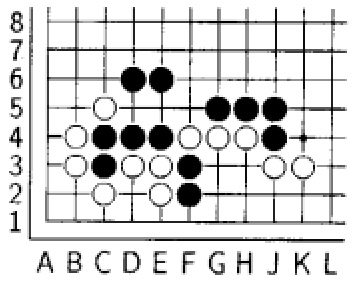
(1) Search

The most important point of this is that GOG is not applied to overall search. The search paradigm just covers a limited search space, adapting it to Chess proportions looks like the limit. Programming a problem with the same or smaller range of search space as reeded for Chess could match the best of human player. Go search is far larger than that of Chess.

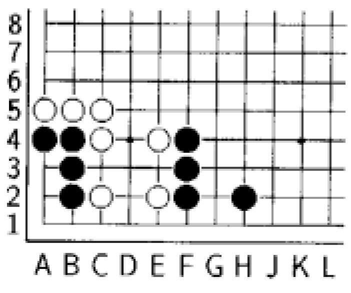
The next point is that as well as accepting the indispensability of search as a procedure of recognition, we also limit the adoption of search to local search with a single purpose and target. This limitation also create some limits. We cannot solve double target and double purpose problems.

A double target problem is shown in Fig.27a. When black places at F1, a string with E3 or F4 might be captured.

A double purpose problem is shown in Fig.27b. When black places at B1, the territory at left corner might be saved or connection between black left and right groups will succeed.



(a) Double target



(b) Double purpose

Figure 27: Future Assignment

The third point is the search routine will only start after the main program breaks a problem down at local search level. This is introduced as pre-procedure which will be positively used before wider search becomes necessary, and which avoids the explosion of combinations.

(2) Ambiguities

Most concepts of Go are quite ambiguous; some are difficult to define. Even if there are some concepts that everyone understands, there are any number of difficulties as to the practical use of those concepts.There is a big difference between reading and understanding a Go book, teaching Go, and playing Go. However, programming Go requires us, unreasonably, to define ambiguous things. Let's look at strength of group as an example. For human beings, the concept of strength is ambiguous, it has qualitative and relative significance; the strength of group is determined by the location of adjacent stones and their relations. So we have to define strength for GOG. For convenience, we defined it as

$$\text{strength} = \text{a} \times (\text{rate of siege}) + \text{b} \times (\text{size of territory})$$

The rate of siege and size of territory are probably the most important primary factors of strength. This function is just equal to the First approximate. So we adjust it for actual use with a consideration about adjacent groups. However it is still pretty far from the real feeling of playing Go.

(3) Exceptions

Dealing with exceptions, at least we have to define the irregular situation and make GOG recognize it. Most irregular situations can be recognized if we define them as "

When a player is at a disadvantage and there is no move which makes the tide turn to his favor with the regular procedure, or when he is attacked at two or more places but there is no move defending these places simultaneously with the regular procedure."

Generally, when we don't notice an enemy attack or we think our enemy is easy to deal with, we often get a nasty surprise. If the situation changes suddenly, it is a sign of irregularity. By using this sign, a move is always forestalled. Therefore we should examine the change of the situation after we wasted one turn.

(4) Cooperation

The ability of systematic integration for Go could roughly be interpreted as the ability to derive a new measure from combination of AJIs.

6 Conclusion

The Go playing system, GOG, has been developed as a part of intermediate stage of the Fifth Generation Computer System Project since 1985. Compared with Western countries, we have less accumulation of such researches. Therefore ICOT pushed this research forward by organizing AI scholars as a working group (CGS-WG) and conducted this research with the Electrotechnical Laboratory of the Agency of Industrial Science, which has reached satisfactory results in this field.

In the first half of 1985, we decided the basic methods of GOG, such as data structures, how to calculate the potential, and so on. We chose to use ESP running on PSI for a test program, then designed a concept of the software module. Also we developed a prototype which has board tools including ATARI, AGEHAMA, territory calculation and so on as main functions.

In the second half of 1985, as the next step, we researched human against human Go games and used that information as data structures. Then we aimed to judge positions and situations. To actualize this, we designed and produced modules for input and output, recognition of positions, and analysis of game situations as parts of the Go playing module which is our system's essential element for experiment. In parallel with this, we designed and produced development tools, such as tools for analysis, search tree analysis, diagram edition, potential tuning, and so on, for experiment.

In 1986, we aimed to actualize a computer playing Go against a human player and reviewed algorithm for capture, enclosure, actual methods of adaptation, and judgment of life and death; this last one was particulary difficult. By the end of the year, a prototype with the above functions and running on PSI had been developed.

In 1987, we improved data structures and knowledge of candidates, expanded functions of them, and intensified cooperation among knowledge. Also, we reviewed the TSUMEGO program, however its responding speed reached a limit after the system's improvement and expansion increased the processes of calculation, so we didn't adapt TSUMEGO into GOG. We adapted this TSUMEGO program as a stand alone system on PSI, and programmed it as a parallel system on a parallel inference computer by way of experiment. Later, we moved it from PSI-I to PSI-II which is a small-sized and accelerated PSI, and tuned it up. Currently, PIM (Parallel Inference Machines) have been under development at ICOT, following this we have been rewriting GOG as a parallel program. As the result of parallel programming, GOG will show us the great increased performance of problem management. Even if our system use a local search at the recognition of positions and a trial-and-error method for cooperation among knowledge or results of recognition, our system could be recognized as the program, which reads just the next move, compared with Chess programs. By increasing its knowledge, polishing it up, or enriching its local search, it is possible to increase

GOG's ability to play Go. However, rapid progress is not expected at all if GOG reads just the next move. The reason for this problem is the excessive transaction time that is necessary to process recognition of positions several moves ahead. For accurate judgement of situations in Go, it is necessary to recognize not only territories but also each stones' life and death or strength. Therefore all transactions which are currently adapted into GOG are necessary for recognition of the positions several moves ahead; if we try to read the entire game, the transaction time will be the time necessary to recognize each position times the number of positions that the game has.

Some other effects of parallel programming are expected. ESP's descriptive ability proved of great benefit in describing knowledge of Go and cooperation among information. Actually, this ability decided our system's structure. Therefore, the new parallel environment could suggest many innovative ideas and techniques. For these reasons, we expect that GOG's Go playing ability will get much closer to human's when we improve the quantity and quality of GOG after rewriting it as a parallel logic program.

7 Acknowledgment

We would like to thank Mr. Yokoi, the former chief of the second research laboratory at ICOT and the current director of EDR, who created the original opportunity to start this research, Prof. Torii at Osaka University, who is the chairman of CGS-WG, and all members of CGS-WG, who have given support and helpful comments. We wish to thank Mr. Hikichi at SRA and Mr. Ojima at KKK, who are members of CGS-TG and have developed a part of GOG, and to thank many people at Mitsubishi Electronic Co. in charge of the joint research works. Also our thanks go to Mr. Okada, the director of ETL, who has always given support. Finally, we wish to thank Mr. Ogiso at FTL for his effort of translating this paper into English.

References

[1] Sanechika,N. "Strategy of a knowledge-oriented Go program Go.1", IPSJ SIG Reports SYM432 (Nov. 1987) (in Japanese)

[2] Sanechika,N. et al. "Notes on modeling and implementation of human player's decision process in the game of Go", Bul.Electorec.Lab.Vol.45(1981)

[3] Sanechika,N. "Programming the Decision Process in Go", Tech. Rep. of IPSJ (JUN. 1982) (in Japanese)

[4] Mano,Y. "An approach to Conquer Difficulties in Developing a Go Playing Program", Journal of Information Processing, Vol.7,No2,July 1984

[5] Reitman,W. & Wilcox B. "The structure and performance of the interim.2 Go Program." Proceedings of the 6th international Joint Conference on artificial intelligence. Tokyo 1979.