ゲ ー ム 情 報 学 8－8
（2 0 0 2． 7． 2 4）

Learning for the Value of Moves by Iteration of Generation of Decision Tree in Go

ABE Nobuharu KOTANI Yoshiyuki
Tokyo University of Agriculture and Technology
{ abe, kotani } @fairy.ei.tuat.ac.jp

Abstract

It is necessary in a game "Go" to limit candidates of moves for searching because the total number of possible moves of a position exceeds 200 on an average. We performed the generation of candidate moves by the decision tree in the past. We got the features around an empty point which could be selected (legal move), using not a pattern but the information about the stone which was in the circumference of selected moves or the coordinates of selected moves from a professional Go player's record. In this paper, first, we generate the decision tree, collect data using generated decision trees, then generate new decision trees. A decision tree is improved by repeating. An evaluation of our method is shown correspondence between our system's optimal candidates and actual moves made by professional Go players. As a result, the number of candidate moves until actual moves appeared was an average of 21.96.

1. Introduction

It is hard to decide a move by only searching in a game Go. The reason is that an average of branch factors of a game tree in Go is more than 200. This is one of reason that computer Go cannot be strong as computer CHESS and SHOGI. And there is a problem that computer Go program cannot understand thinking of human player. But it is possible that a computer generate promising candidate move without understanding thought of human player [1]. We got simple features on a board as data from professional Go player's records and performed learning the value of moves using decision tree in past [2]. This method could generate candidate moves without complex calculation and huge pattern database. We focused static evaluation from opening to middle game.

In this paper, we use decision tree already generated to collecting data and generate new decision tree. We use both of them to selection candidate moves. Moreover we use them to collecting data and generate new decision trees … repeating this way, we try to improve candidate move generated by decision trees.

2. Learning for the moves using decision tree

It is the method that set some information to measure the value of a move and the information collected from professional player's records generates a decision tree. An evaluation is corresponding between optimal candidates by the output of decision tree and actual moves made by professional players.

2.1 Learned Moves

When human plays Go, they consider various things in deciding move. For example, if capture opponent's large stones, if escape friend's stones which is likely to be captured, if increase friend's territory or decrease opponent's territory, and so on. Their purpose is clearness. While, the purpose of such moves that is attack move, defend move, KESHI, KIKASHI, etc. does not be recognized clearly.

Computer can understand former moves using capture search and so on. But, because of deferent opinion between people about later moves programming is hard. So for move, we make computer convert the move: "cannot know what attempt but, quality as the move is high" into value of move. This will enable to generate promising candidate move without attempt or intent of human. In this paper we particularly consider from opening to middle game.

2.2 Information of circumference

When human player decide a move, they consider circumferences of the move. For instance, where many opponents' stone is placed around they select a move that tend to life. By contrast where many friends' stone is placed, they select a move that attack hardily. If there are friend' stones placed in line 1, three-space jump is possible and so on. If computer have enable to understand these circumference, it seems that it can play the move as same as human player. In this paper, for this circumference we use a coordinate of move and the information of stones around the move.

We deem stones on all sides to be stones around the move. We do not think diagonal direction since relation with lengthwise and crosswise is taken more important than relation with diagonal direction. When the four-quarter is seen, have one width. For stones on all sides, we use one unit of stone that is string (a set of adjacent stones of one color) connected by KOSUMI (diagonal). We call this unit "KOSUMI-STRING" [3]. Figure 1 shows information about KOSUMI-STRINGS. (o) relation is shown in Figure 2. Other information is shown in Figure 3, and we treat these information as circumferences.

Necessary description to information is the following. (e) discerption point : when there is a connected point by KOSUMI and one point is occupy by opponent's stones, other empty point is discerption point. It is concern with possibility disconnection. (g)〜(k) around : space to two Manhattan distance from each stone which construct KOSUMI-STRING. (l)〜(n) height : distance from border. In 19-by-19 board height is 1 to 10. (3) Potential value: a kind of influence that each stones radiate.

Furthermore we do not consider life-death for KOSUMI-STRING. Because life-death judging is impossible when too many stone or too few stone. Considering Go's symmetry, a board is divided into eight.

(a) Color of stones
(b) Num. of stones
(c) Length
(d) Width
(e) Num. of discerption point
(f) Num. of liberty
(g) Num. of illegal point for opponent
(h) Num. of around empty point
(i) Num. of around friend's stone
(j) Num. of around opponent's stone
(k) Num. of around edge
(l) Maximum height
(m) Minimum height
(n) Medial height
(o) Relation with move

Figure 1：Information of KOSUMI-STRING

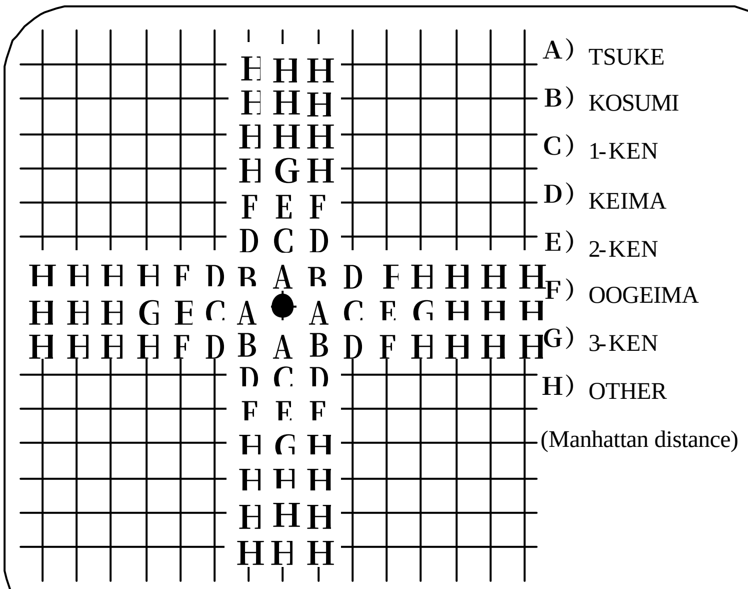

Figure 2 : (o) Relation with move

(1) turn (2) height of point
(3) potential value (4) number of plies

Figure 3: Other information

2.3 Learning of decision tree

To make computer understand the circumference shown chapter 2.2, we get huge data and generate decision tree. The algorithm of the generation of decision tree used in this paper is most fundamental algorithm ID3 [4]. Since it is not related to an essence of this research even if other algorithm is not used, we use more simple it. The information about the circumference offered chapter 2.2 is attributes of decision tree and whether selected move or not selected move is a class of decision tree. We collect such data from professional player's records.

At decision tree, when classes of sub set classified by a query are sameness it become a terminal. But in this research it will suppose that it is hard for sub set to be classified one same class actually, so we define sub set as the terminal when number of factors of sub set become less than a constant number and make a rate of "it is selected move" an output at this case. This output value is the value of move.

In Go, an average of number of the legal move is more than 200, and is more than 300 in opening. So a difference between the data of "selected move" and the data "not selected move" is large and generated decision tree is lopsided. So number of data of "not selected move" must be limited at collecting data.

A selection of the candidate move by generated decision tree is below routine for a position.

(a) for all legal move collect data described chapter 2.2
(b) thread decision tree with each data and get the output value
(c) candidate moves are sorted from what has a high output value

3. Iterative generation of decision trees

At collecting data for generate decision tree, not selected move are collected at random. This is why data of good move and worse move are distributed moderately, and generating good decision tree is possible. In this paper, using decision tree already generated at collecting data we try to generate better decision tree.

Because of a moderate data distribution, decision tree generated by random collected data have a few performance. To get the candidate move by this decision tree as a training data, it assume that better learning becomes possible. We suggest the method that using decision tree already generated to learning again. The routine is shown below.

(i) read records and get data at random

(ii) generate decision tree (0)

(ii) read same records and get data by the candidate move outputted by decision tree (0)

(iii) generate decision tree (1)

This decision tree (1) is seemed that can select better candidate move from it bye decision tree (0). So using both of decision tree (0) and decision tree (1), generation better candidate move will become possible. The routine of candidate move by these two decision trees is below.

(a) get data for all legal move

(b) take each data to decision tree (0) and (1), and add each output value.

(c) the candidate move are sorted what has a high sum value.

Generate decision tree (3), (4)... iteratively this routine(Figure 4)

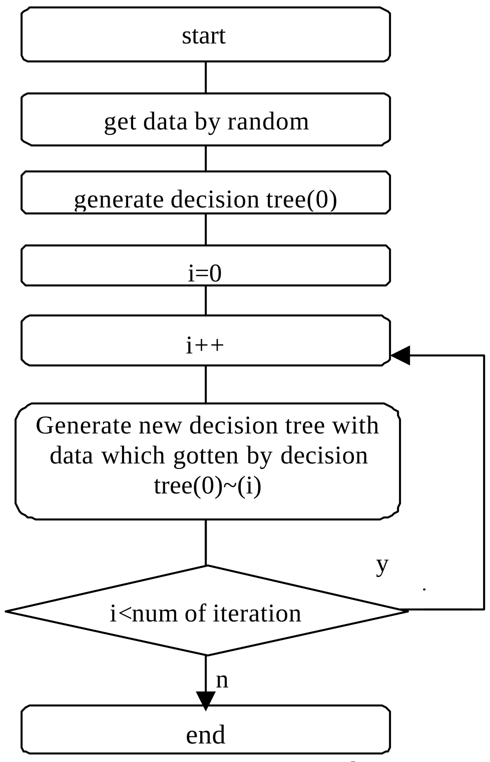
start
get data by random
generate decision tree(0)
i=0
i++
Generate new decision tree with data which gotten by decision tree(0)~(i)
y
i<num of iteration
n
end

Figure 4 : iteration generation decision tree

4. Experiment of performance

We collect data from professional player's records [5], generate decision tree iteratively and evaluate a performance actually. In this research we consider only 1~50 move where from opening to middle game.

4.1 Pre-experiment

Before generating iteration decision tree by the method observed chapter 3, we examine how does decision tree have a capability when collecting data at random. We try to generate decision tree buy several kinds of number of data that is "not selected move" and evaluate the performance. Following each setup of decision tree.

• number of plies	：1〜50
• number of "not selected move"	：10、20、50
• number of records	：776、776、640
• number of factor at terminal	：200
• maximum depth	：12

Table 1 shows result of the pre-experiments. We used 50 training records and test records for evaluation. We read records from 1 to 50 plies, thread decision tree with data from all legal move at every move, and decide the candidate move from the high output value in order. We evaluate by number of candidate move until it is corresponded with selected move of record and rate whose number of candidate move until it is agreement with selected move is less than 20. Judging from result of test-data at each setup mainly, we decide that number of "not selected move" is 50.

Table1：result of pre-experiment

Not selected move	Num. of record	Num. of to selected move		Rate of num. of candidate move is less than 20	
		training data	test data	training data	test data
10	776	22.274	26.024	0.6602	0.6208
20	776	23.288	27.173	0.6466	0.6140
50	640	21.961	24.719	0.6504	0.6364

4.2 Experiment

We generate iteration decision tree using above setup, and compare performance of iteration decision tree with it of not such tree.

4.2.1 A setup of a decision tree

In this experiments, each setup of iteration generation decision tree is followed. To ask whether this method depend on number of data we try to examine four kinds of record number.

• number of plies	：1〜50
• number of not selected moves	：50
• number of record	：80、160、320、640
• number of factor at terminal	：200
• maximum depth	：12
• number of iteration generation	：1〜20

4.2.2 The way of evaluation

We take all legal move's data to decision trees every position of records (for training/test) and make a move whose data has a high output value the candidate move in order. And we evaluate it by the number of candidate moves until it is corresponded with the selected move, rate of it, and rate of it is less than 20. Note that the value of move is sum value of the output value by decision trees when generating decision tree iteratively.

4.3 Results

Table 2 shows the result of the experiment when the number of iteration of generation is set 3 and four kinds of record number. Way of the evaluation is number of candidate move.

Table2 : Number of candidate moves until it is corresponded with the selected move

Record / Iteration	Training data				Test data			
	80	160	320	640	80	160	320	640
0	25.72	24.366	20.999	21.961	39.29	37.446	30.718	24.719
1	22.881	22.875	22.659	55.11	41.284	41.648	26.397	60.334
2	21.302	17.906	24.026	50.518	39.594	37.252	29.071	55.783
3	20.388	15.929	19.35	42.382	38.634	36.551	28.206	48.454

The performance of the setup whose number of records is 640 is best when number of iteration is 0. But as a result of iteration, record numbers 320 has best performance, so we continue to examine using the setup whose number of record is 320. Result is shown at Figure 5~8. Figure 5 and 6 show number of candidate moves until it is corresponded the selected move.

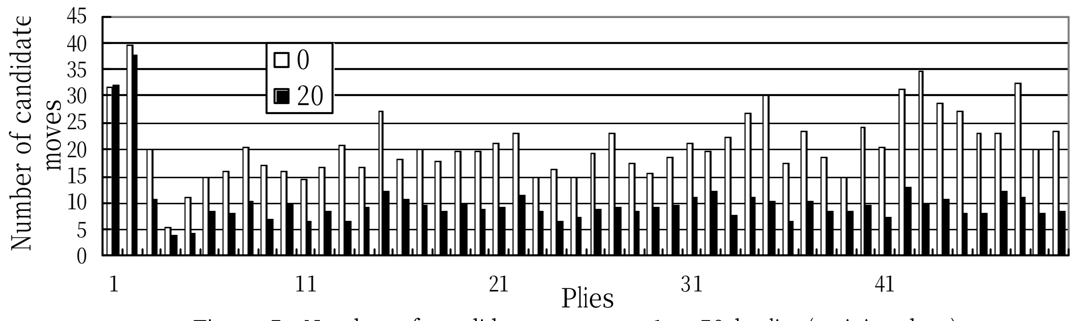

Figure 5 : Number of candidate moves at 1st-50th plies(training data)

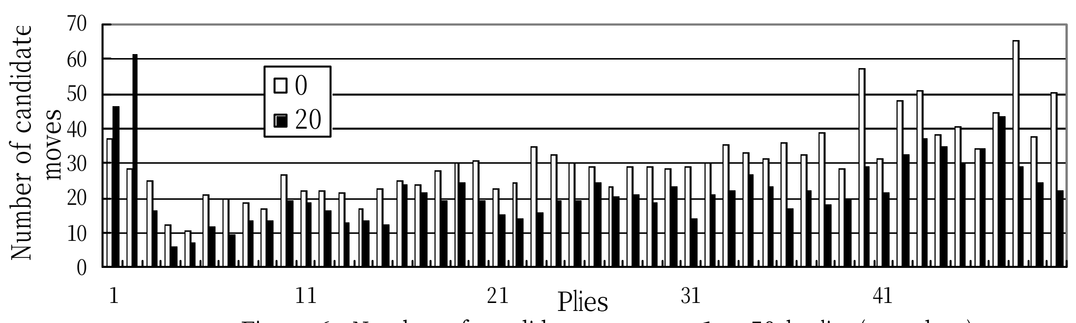

Figure 6 : Number of candidate moves at 1st-50th plies(test data)

Figure 7 and 8 show rate of the selected move appear at n-th candidate moves.

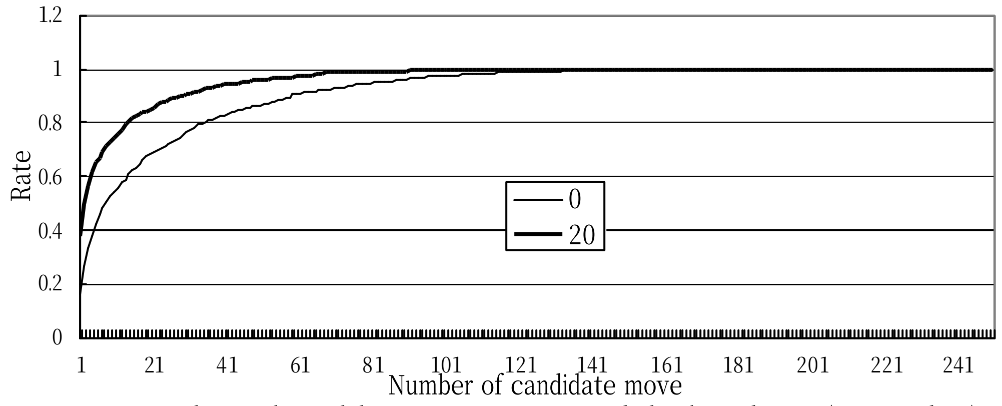

Figure7 : rate that n-th candidate move is coressponded selected move(training data)

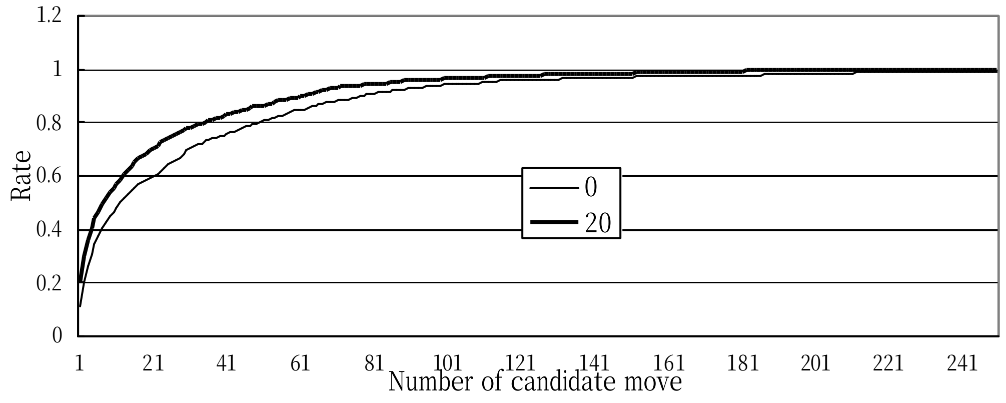

Figure8 : Rate that n-th candidate move is coressponded selected move(test data)

Table 3 illustrate result of **number of the candidate moves until it** is corresponded with selected move, rate of it, and rate of it is less than 20 at iterations 0 ~ 20.

Table3 : results of examination at each iterations

Iteration	Num. of candidate move		Rate that num. of candidate move is less than 20		Iteration	Num. of candidate move		Rate that num. of candidate move is less than 20	
	Training	Test	Training	Test		Training	Test	Training	Test
0	20.999	30.718	0.6852	0.5940					
1	22.659	26.397	0.6652	0.6128	11	11.891	22.941	0.8300	0.6860
2	24.026	29.071	0.6468	0.5892	12	11.647	22.514	0.8348	0.6908
3	19.350	28.206	0.7424	0.6404	13	11.288	22.424	0.8440	0.6900
4	17.775	26.411	0.7576	0.6508	14	11.162	22.566	0.8392	0.6840
5	14.821	24.803	0.7756	0.6536	15	11.339	22.537	0.8292	0.6832
6	14.077	23.926	0.7792	0.6604	16	10.364	22.407	0.8492	0.6868
7	13.164	23.380	0.7996	0.6660	17	10.326	22.351	0.8500	0.6876
8	12.830	23.124	0.8036	0.6672	18	10.235	22.318	0.8480	0.6888
9	12.226	22.741	0.8144	0.6672	19	10.136	22.127	0.8480	0.6912
10	13.604	23.912	0.8172	0.6752	20	10.016	21.964	0.8504	0.6928

5. Discussions

Our method is successful by table 3 expressed that the more number of iteration generation decision tree increase, the more performance of candidate move increase. Especially increase of the accuracy of the candidate move at training data is distinguished. We assume that it is because our system gets data from same records repeatedly. Performance of the final result exceed it of the setup that number of records is 640 and iteration is 0, so using this method it is possible to learn enough with few number of record or number of data.

Best advantage is appeared at 4th or 5th plies recognized Figure 5 and 6. 4th or 5th plies are often the move that occupy last corner points, so it is assumed that our system can learn easily. As number of plies increase decrease of the performance is distinct at iteration 0, but at iteration 20 it is not so.

The experiment of 640 records cannot learn well. This is thought that its cause is the setup of decision tree. It seems that requirement that the sub-set of the data become the terminal does not depend on number of data. However comparing the performance of first decision tree that the data collected at random it is proportionate to number of records, so other cause is exist. As its performance is increase in increase of iteration, if more experiment it may be success as the setup that number of record is 320.

From Figure 5 and 6 it is illuminated that number of candidate moves is increase and decrease in some set of move. It may express a period of episode.

Table 3 observes that increase of performance is exiguity at over iteration 15. Because that the more increase of decision tree, the more increase of time for the generation candidate move, it may be suitable to stop the generation newly decision tree at temperate number.

6. Conclusions and Future Works

We have suggested that we regard circumferences of moves as features of moves and perform learning for the values of moves using decision trees. Using ready-made decision trees when collecting data, we generate new decision trees 20 iteratively. The final decision trees generate candidate moves at 1st-50th moves in professional player's records. As a result, the number of candidate until professional's selected moves are emerged is 21.96 on an average and the rate of them is less than 20 is 69.28%. This result is higher performance than candidate moves by decision tree generated by random collected data

We will try to experiment in different setups of decision trees be independent of the size of data. And we introduce information of the last move as a new attribute of decision tree. Since in Go, it is pointed that the moves which are near the last moves are evaluated better, this information is effective as pauses of episodes. We also try to use information of player's last moves and expect expressions of consecution of moves. From these, perhaps the expression of a book may be possible.

To evaluate our method, the next move problem, the result of playing system, and so on are considered.

References

[1] Sei Shinichi, Kawashima Toshiaki : The Experient of the Go-program "KATSUNARI" using Memory-Based Reasoning, Game Programming Workshop '96, pp.115-122, 1996

[2] Abe Nobuharu, Kotani Yoshiyuki : Learning for the value of moves using Decision Tree in Go, Game Programming Workshop '01, pp.156-159, 2001.

[3] Abe Nobuharu, Kotani Yoshiyuki : Learnng the Strength of Strings using Decision tree in Go, Index SIGNotes Game Informatics No.005, pp1-8, 2001.

[4] J.R.Quinlan : Induction of decision trees, *Machine Learning*, 1, 81-106, 1986.

[5] BGA Software Catalogue, http://www. britgo.org/gopcres/gopcres1.html