JAIST Repository
https://dspace.jaist.ac.jp/

Title	Detection and Labeling of Bad Moves for Coaching Go
Author(s)	Ikeda, Kokolo; Viennot, Simon; Sato, Naoyuki
Citation	IEEE Conference on Computational Intelligence and Games (CIG2016): 1-8
Issue Date	2016-09
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/14266
Rights	This is the author's version of the work. Copyright (C) 2016 IEEE. IEEE Conference on Computational Intelligence and Games (CIG2016), 2016, 1-8. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Detection and Labeling of Bad Moves for Coaching Go

Kokolo Ikeda[1], Simon Viennot[1], and Naoyuki Sato[1]

[1]School of Information Science, JAIST, Japan

Abstract:
The level of computer programs has now reached professional strength for many games, even for the game of Go recently. A more difficult task for computer intelligence now is to create a program able to coach human players, so that they can improve their play. In this paper, we propose a method to detect and label the bad moves of human players for the game of Go. This task is challenging because even strong human players only agree at a rate of around 50% about which moves should be considered as bad. We use supervised learning with features largely available in many Go programs, and we obtain an identification level close to the one observed between strong human players. Also, an evaluation by a professional player shows that our method is already useful for intermediate-level players.

I. Introduction

Computer programs to play games have improved a lot this last decade, with the use of machine-learning and Monte-Carlo Tree Search (MCTS). The last success was the defeat of a professional Go player by AlphaGo in 2016[6], by using a combination of deep convolutional neural networks and MCTS. The strength of other programs for the game of Go is now expected to increase quickly, so we can consider that creating a strong Go program is not as challenging as before.

However, there are still other difficult and interesting problems for computer intelligence in the area of games, especially the game of Go. A first one is the creation of entertaining programs. Computer programs are still too frequently boring for human players, because they tend to use similar strategies repeatedly. Another interesting problem is the ability to coach human players, by showing them their mistakes, and explaining them how to improve.

For entertainment or coaching purposes, programs need some new abilities usually not considered when strength is the only target. For example, for entertaining humans, a control of the position is needed, so that both players keep a reasonable chance of winning. It can be achieved with intentionally gentle - but natural - moves. The thinking time used for each move or the resign timing are also important. For coaching humans, bad moves need to be detected, and some explanation is also needed, either with figures, text or speech.

kokolo@jaist.ac.jp
sviennot@jaist.ac.jp
satonao@jaist.ac.jp

In this research, we consider the problem of coaching Go players. In the case of the game of Go, it is frequent for human players to review their own games with a stronger player and seek advice about which moves were bad. So, an ideal coaching computer program should be able to detect the bad moves, to label them with the type of mistake, and finally to give a more detailed explanation. Also, a figure showing a better move with its consequences would be useful. In this research, we consider only the problem of detecting and labeling the bad moves.

Deciding which moves should be considered as bad is a challenging task. In a preliminary experiment, we asked strong players to show the bad moves in game records of intermediate-level players. The strong players only agreed at a rate of around 50%. Also, we will show that a naive approach like using only the drop of winning ratio from the point of view of a strong computer program does not work well. Many bad moves from the point of view of humans are locally non-optimal moves (for example a bad shape), but the loss in terms of winning chances is in fact small. In this paper, we propose to use machine-learning to address this problem.

In Section II, we give some more details about coaching Go, and how it is usually done between humans. In Section III, we discuss some related work. Then, in Section IV, we describe our approach based on supervised machine learning. Section V describes our main experiments, with a machine learning for detecting bad moves, and a separate machine learning for labeling them. The result is evaluated with a professional Go player.

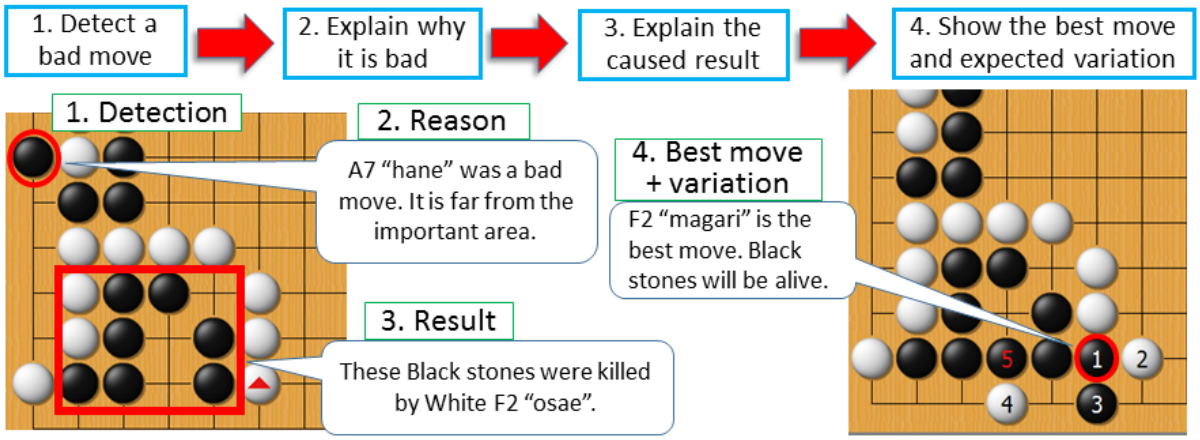

Fig. 1. Complete process for helping players to correct their bad moves.

II. Coaching Go

Go is an ancient game, especially popular in Asia, with a rich variety of sub-problems and strategies. For this reason, many Go players find their satisfaction not just in playing the game, but in trying to improve their play and becoming stronger.

There are many ways to improve at the game of Go, such as replaying professional games, solving local life and death problems (tsumego), or reading books about common tactics and patterns. But it is often considered that one of the best way to improve is to play a game with a stronger player, and to review the game with him.

Coaching Go (Shido-Go in Japanese) is a special type of game, where amateur players pay some professional or semi-professional player to play and review a game. There is a strong demand from amateur players for such games, but it can be expensive. Also, intermediate-level players are often reluctant to invest the money or the time in such coaching games, because they feel that their level is still too low for that. If a computer player could perform the same kind of coaching, it would be of great help for many amateur players, especially from beginner to intermediate level.

We surveyed in Go clubs in Japan how strong players teach intermediate-level players about their bad moves. It usually follows the process shown in Figure 1. First, a bad move is detected (1). Some reason (2) is given on why it is a bad move. We call this step "labeling" in this paper. Then, a more detailed explanation is given on what happened as a result (3) of the bad move. Finally, a better move is shown, with the expected best variation (4).

In this paper, we address only the first two steps, i.e. the detection of bad moves, and the labeling with some "reason". Our goal in the future is to create a computer program able to perform all 4 steps.

III. Related Work

Entertaining and coaching players is a developing area of research on board games. In 2013, Ikeda et al. (authors of this paper) proposed a computer Go program able to entertain players by using various strategies and controlling the board position [3]. In 2015, Kameko et al. used machine-learning to generate comments in natural language about Shogi positions [4]. Also in 2015, Ikeda et al. (authors of this paper) used machine-learning to learn the natural language names usually used by humans to refer to moves in the game of Go [5]. This is an important part for a coaching Go program, since moves are usually refered by shape names and not by coordinate positions in the game of Go. For example, in Figure 1, A7 is called "Hane".

IV. Approach

The ultimate goal of our research is to make a coaching computer player who plays gently against human players, corrects bad moves and explains how to think/play. It would encourage human players to continue playing while their skills are improving. As a first step towards this goal, we try (1) to detect bad moves from a game record and (2) to associate an explanation label to each bad move. Then, the computer player could ouput something like "The 17th move at D4 was not good, because the local shape is bad. D5 is better."

The definition of "bad move" is not trivial. If the set of best moves can be defined and calculated, though theoretically it is possible, it may be possible to say that the other moves are all bad moves. However, for

coaching intermediate players, usually only *fairly* bad moves are pointed out because such players will be confused or depressed if too many moves are corrected.

For selecting *fairly* bad moves, it will be effective to refer to the "winning ratio" computed by computer players. Now, we assume that White is played by an MCTS player, and Black is played by an intermediate human player. An MCTS program calculates not only the next best move but also many statistics such as the expected winning probability (winning ratio). When the ratio for White is increased, for example from 30% to 50%, it means that Black played a fairly bad move which loses a big advantage, 20%. Usually, a Black move which loses 20% winning ratio should be pointed out, before any other move that loses only 2%.

But it should be noted that winning ratio will be not sufficient to select bad moves for effective coaching. Human teachers often point out and correct some kind of bad moves even when the loss of winning ratio is not so serious. One example is shown in Fig. 2. The shape of Black move A is bad, and B should be played. We think almost all Go teachers will point out this move, even if the difference of winning ratio of A and B is only about $1-2\%$. To detect such a move, "shape goodness" should be computed and referred. Another example can be considered in end games. Assume that Black is almost surely winning, and the territory advantage is 12 points. When Black played a bad move which loses 4 points, it will be pointed out, even if the winning ratio is only slightly changed from 99% to 98%. To detect such a move, "territory advantage" should be computed and referred. So, we calculate and use several values as input features, for accurate detection of bad moves. The employed features are explained in V-A.

The detection and labeling of bad moves are done seperately. The whole procedure is as follows:

1) Many handicap games are done. Game records are collected.
2) Bad moves are selected by strong human players, with only 5 to 10 moves selected per game.
3) Also, one type (why the move is bad) is labeled on each bad move, from some candidates.
4) Many features are calculated by a computer program, for each move. We obtain a set of items (feature1, feature2, ..., bad/good, type).
5) A supervised learning is executed by using all items where "bad/good" is the output. The result is the "detection system".
6) Another supervised learning is executed by using "bad" items where type is the output. The result is the "labeling system".

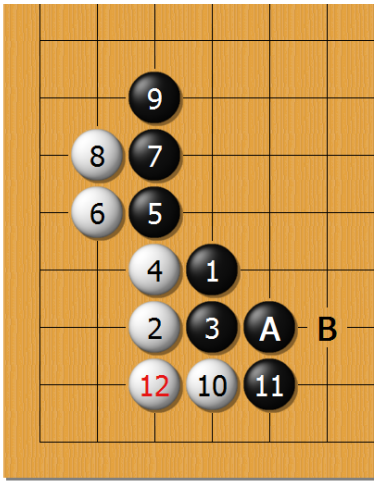

Fig. 2. An example of bad shape, A is bad, B is good. This A will be pointed though the loss of winning ratio is not so big.

V. Experiments

In this section, we show four series of experiments. The brief content is as follows:

1) Preliminary experiment to show that the winning ratio is not a sufficient feature for detecting bad moves
2) Learning of bad move detection system, and comparison with human's decision
3) Learning of bad move labeling system, and comparison with human's decision
4) Evaluation of the detection and labeling systems by a professional player

A. Preparation

As described at the end of Section IV, we need to gather many handicap games, to let strong human players select bad moves and label them, and to calculate many feature values for each move. We employed our computer Go program "Nomitan". It is ranked 3d on the KGS server, which is not so strong, but not so weak. First, we asked 8 intermediate-level human players (from about 7k to 1d) to play against Nomitan, using a 13×13 board and with 2 to 4 handicap stones, as they want. Totally, 108 games were collected.

Next, we asked three strong human players (about 4d to 7d on KGS) to select bad Black moves and select a type label for each bad move. It was requested to ignore bad White moves, and to select about 5 to 10 bad moves per game. A type label for each bad move is a brief reason explaining why the move is bad. It was selected from the following 10 candidates. Since some types were rarely labeled, they are integrated into 5 groups.

- **Group-1**
  - Local shape is bad.
- **Group-2**
  - Gain is small.
  - The move is too defensive or fearing a risk, then the gain is small.
- **Group-3**
  - The move is far from the hot area.
  - The move is far from the hot area, White stones should be attacked.
  - The move is far from the hot area, Black stones should be defended.
- **Group-4**
  - The player seems to do a reading mistake (i.e. a tactical error when considering what happens a few moves ahead).
- **Group-5**
  - The move is too passive. It seems to be only responsive to the last White move.
  - The move helped White stones to be stronger.
  - Other reasons.

For 102 in 108 games (set denoted by G_{102}), only one of the three human players did this selection and type labeling of the bad moves. For 6 games (set denoted by $G_{common6}$), 244 Black moves, all of the three human players did this work. This allows us to compare the selection result between humans. For example, Table I shows the difference in bad move selection between two strong players A and B.

TABLE I
BAD MOVE SELECTION BY STRONG PLAYERS A AND B

	B good	B bad
A good	180	27
A bad	24	13

Out of 244 moves, player A selected 37 moves as bad moves, but only 13 of these 37 moves are also selected by player B. This result shows that bad move detection is not a simple work even for strong human players.

Totally, 4836 Black moves were collected from 108 games. For each of these moves, we calculated 29 feature values to be referred in supervised learning. Here, some important features are explained, and the other ones are explained in Appendix A. Please note that such features are not specific to our program. They can be easily calculated by most MCTS programs.

- **handi**, the number of handicap stones.
- **move**, the number of moves played.
- **wrbefore, wrafter, wrdiff**, expected winning ratio before the move, after the move, and its difference.
- **trbefore, trafter, trdiff**, expected territory advantage before the move, after the move, and its difference.
- **shaperate, shapelog**, shape goodness calculated by Bradley-Terry model [2], relative value and absolute log value.
- **dist1b**, Euclidean distance between the actual Black move and the estimated best move.
- **ownbefore, ownafter, owndiff**, ownership of the position before the move, after the move, and its difference. High ownership means that the area is occupied by Black, i.e. the Black stones in the area are strong, or the White stones in the area are weak.

B. Preliminary Experiments: feature selection for good/bad detection

In this section, binary supervised learning experiments about "detection system" are shown, to prove that many features should be used for detecting bad moves accurately.

We have 3963 good move instances and 873 bad move instances. Since such unbalance among the numbers of instances is not preferable in classification, 2000 good move instances are randomly removed in this experiment. Since there are a lot of candidate methods for binary classification, we employ Multilayer Perceptron in a free machine learning platform, Weka version 3.6.11 [1].

For evaluating the performance, the F-measure, the mixed value of precision and recall, is used. For example, in the case of Table I, if we assume that the decision of B is always true, the precision of A about bad moves is $\frac{13}{24+13} = 0.351$, the recall of A about bad moves is $\frac{13}{27+13} = 0.325$, and the F-measure about bad moves is $\frac{2 \cdot 0.351 \cdot 0.325}{0.351+0.325} = 0.338$.

When only **wrdiff** (how the winning ratio is changed by a Black move) is used as the input, the F-measures are 0.812/0.299/0.654 (F-measure about good moves / F-measure about bad moves / weighted average for good and bad moves). These values are the averages of 10-folding validation. It is not strange that the F-measure about good moves is better than that about bad moves, because the number of good move instances (1963) is still bigger than the number of bad move instances (873).

We tried to improve the performance by adding other features. Table II shows the result. By adding one or two features, the total F-measure is increased by 0.008 to 0.031. It is clear that shape goodness and territory advantage should be considered for accurate detection. Maybe it is interesting to see that **move** or **wrbefore/wrafter** fairly improve the performance. This is because usually bad moves in early stage frequently affect the game

consequence, and bad moves after losing game (for example winning ratio is under 30%) are not pointed by human coaches.

Finally, when using 9 features **wrdiff, wrbefore, wrafter, shapelog, trdiff, trbefore, trafter, move, owndiff**, the F-measure about bad moves is significantly improved from 0.299 to 0.444. We can conclude that, not only the winning ratio, but also many other features are needed for an accurate detection of bad moves.

TABLE II
GOOD/BAD DETECTION RESULTS. USED FEATURES AND F-MEASURES

features	F-measures	gain
wrdiff only	0.812/0.299/0.654	-
+wrbefore, wrafter	0.815/0.361/0.675	0.021
+shaperate	0.814/0.326/0.664	0.010
+shapelog	0.812/0.357/0.672	0.018
+trdiff	0.809/0.381/0.677	0.023
+trbefore, trafter	0.817/0.389/0.685	0.031
+handi	0.810/0.333/0.663	0.009
+move	0.812/0.378/0.678	0.024
+dist1b	0.813/0.322/0.662	0.008
+owndiff	0.812/0.330/0.664	0.010
+8 features	0.826/0.444/0.709	0.055

C. Machine-Learning for Detection

In Section V-B, we observed that 8 additional features are effective to improve the detection accuracy, and there 10-folding self validation is used. In this section, the learning set and the test set are manually separated, and the performance for test data is compared to the performance between human strong players.

As shown in Table I, decisions are fairly different from each other, even among strong players. Table III shows F-measures of each player for another player, we can see A for B is relatively far, B for C is relatively similar. The simple averages are **0.892/0.435/0.820**. We try to achieve these values by machine learning.

TABLE III
F-MEASURES OF GOOD/BAD DETECTION

	F-measures
player A for B	0.876/0.338/0.794
player B for C	0.907/0.525/0.844
player C for A	0.895/0.442/0.821
average	0.892/0.435/0.820
MP for player A,B,C	0.875/0.409/0.800

$G_{common6}$ is used as the test set, including 609 good moves and 117 bad moves. G_{102} is used as the training set, including 3354 good moves and 756 bad moves. In order to balance the numbers of good moves and bad moves, we clone each bad moves of the training set from one to three, then 3354 good moves and 2268 bad move instances are used for training. We think this cloning method is better for obtaining a good detection system than deleting 2000 good moves, but it should be noted that 10-folding self validation becomes unfair when using this cloning method, then another way was used in Section V-B.

Multilayer Perceptron (MP) in Weka is used, and the same 9 features shown in Section V-B are referred. The achieved F-measures were **0.875/0.409/0.800**. They are slightly worse than the average among strong human players, but better than those of player A for B. We can guess that the decisions (detected bad moves) are not so strange compared to those from strong human players.

D. Machine-Learning for Labeling

The second step is to label a type on each detected bad move. We have totally 873 labelled (bad move) instances, the numbers of instances of 5 groups are 228, 228, 212, 98 and 107.

Like the experiments shown in Section V-B, we did some preliminary experiments to select a classification method and select the referred features. After comparing several methods available in Weka, such as J4.8, LADTree, SMO or Multilayer Perceptron, we selected "Logistic" as the classification method.

The total F-measure (averaged by 10-folding self validation) is 0.406 when using the full set of 29 features. We tried to improve the F-measure by removing some features to avoid overfitting. In almost all cases the F-measure is decreased by removing features, this suggests that more complex features are effective in the labeling system compared to the detection system. The F-measure is slightly increased when removing some of 7 features, and finally the F-measure is 0.434 when removing all the 7 fearures.

Next, as in the experiments shown in Section V-C, we separated 873 instances in a learning set and a test set, and compared the F-measure to that among human players. The learning set contains 756 instances from G_{102}, which are selected as bad moves. The test set contains 69 instances from $G_{common6}$, which are selected as bad moves, **by two or three of strong human players A,B,C**. 48 instances of $G_{common6}$ are selected as bad moves by only one of the three players, then it is impossible to compare whether the labeled types are the same or different.

The total F-measure among human players is shown in Table IV. The averaged F-measure is 0.483, which means that two players frequently gave different labels to a bad move. The achieved F-measure by Logistic is 0.499, this

is better than the average. Since the number of test set is only 69, we think this is just a lucky case. In fact, when using other sophisticated classifiers, the F-measure is only in the range from 0.35 to 0.42. The labeling system (the second step classification) will be more difficult than the detection system (the first step classification), because the number of output classes is bigger, 5 instead of 2, and because the size of the learning set is smaller, 756 instead of 4110. We guess the performance will be fairly improved when increasing the size of the learning set.

TABLE IV
F-MEASURES OF BAD MOVE TYPE LABELING

	F-measure
player A for B	0.482
player B for C	0.436
player C for A	0.531
average	0.483
Logistic for player A,B,C	0.499

E. Evaluation by a Professional Player

In Sections V-C and V-D, mainly F-measure values are used for evaluation, and they are compared to the average F-measure between strong human players. However, F-measures cannot evaluate whether terrible decisions exist or not, for example "a really bad move is not detected" or "the definitely-best move is detected as a bad move". Then, an absolute evaluation by a professional player is done.

$G_{common6}$ are the games that all three strong human players select bad moves and label their types. At the first, the detection system (Multilayer Perceptron) employed in Section V-C was used for $G_{common6}$, then 46 bad moves were selected. Next, the labeling system (Logistic) employed in Section V-D was used for these bad moves, then we obtained 6 game records where bad moves are selected and labeled. A game labeled by our method is shown in Appendix B.

Totally 24 game records were sent to a 6d professional player in a blind manner, and we asked him to give a score for each game record, about (1) How well the bad move detection is done, and (2) How reasonable the type labeling is done. The scoring criterion we asked was as follows:

- 100 points: at the same quality of human professional coaches
- 90 points: at the same quality of human 6d amateur coaches
- 70 points: there are some problems, but still sufficiently valuable for intermediate players.
- 50 points: there are many or serious problems, then not so valuable even for intermediate players.

Table V shows the points given for (1) bad move detection, and Table VI shows the points given for (2) type labeling. The average scores of players A, B and C are similar, about 80 points, but individual scores are not so stable, from 60 points to 100 points. Please note that 90 points are not achieved even though they are about 6d amateur players. The average scores of our systems are worse than that of strong human players, by about 6 points, but better in some games. Total average 74.2 and 76.7 are not bad, clearly better than 70 points level, "sufficiently valuable for intermediate players".

We consider that our method is promising or even already useful, and we can expect the performance to improve if we collect more games as training data.

TABLE V
EVALUATION SCORES BY A PROFESSIONAL, FOR BAD MOVE DETECTION

handicap	4 stones		3	2 stones			
game ID	1	2	3	4	5	6	average
player A	75	*60*	90	85	90	90	81.7
player B	90	75	80	75	75	90	80.8
player C	80	85	90	*65*	85	80	80.8
our method	70	70	80	75	70	80	74.2

TABLE VI
EVALUATION SCORES BY A PROFESSIONAL, FOR TYPE LABELING

handicap	4 stones		3	2 stones			
game ID	1	2	3	4	5	6	average
player A	70	70	80	90	80	90	80.0
player B	90	75	80	70	75	100	81.7
player C	95	95	70	70	90	85	84.2
our method	70	70	90	80	70	80	76.7

VI. CONCLUSION AND FUTURE WORK

Since strong computer players can be implemented for many games, entertaining and/or coaching computer players have become a new target of computer intelligence. In this paper, we design a system for detecting and labeling bad moves played by human players, in the game of Go. It was shown that these works are not simple nor easy, there was around 50% mismatch even among strong human players, and many input features are needed for making adequate decisions. We collected 4110 moves labeled by strong players, calculated 22 features, and employed two-step supervised learning. The qualities of detection and labeling were evaluated by a professional Go player, it was shown that both of them were clearly at a useful level, though slightly worse than the level of strong human players.

As future work, the number of learning data should be increased because these supervised learning problems

are a difficult task. Many features are needed and then much learning data is needed to avoid overfitting. Also, we want to try other tasks for coaching. Especially in the case of correcting bad moves in the game of Go, it is preferred after detection and labeling to explain the result of each bad move, and to show the best move with its consequence. Playing various games with an understanding of abstract concepts is now not such a difficult task for computer intelligence, but coaching human players with an explanation of such abstract concepts is still a challenging task.

REFERENCES

[1] J. R. Quinlan, C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993

[2] Remi Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go", *ICGA Workshop*, (2007)

[3] Kokolo Ikeda and Simon Viennot, "Production of Various Strategies and Position Control for Monte-Carlo Go - Entertaining human players", Proceedings of the 2013 IEEE Conference on Computational Intelligence and Games (CIG), pp. 145-152 (2013)

[4] Hirotaka Kameko, Shinsuke Mori and Yoshimasa Tsuruoka, "Learning a Game Commentary Generator with Grounded Move Expressions". Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG), pp. 177-184 (2015)

[5] Kokolo Ikeda, Simon Viennot and Takanari Shishido, "Machine-Learning of Shape Names for the Game of Go", 14th International Conference Advances in Computer Games, (2015)

[6] David Silver, Aja Huang et al. "Mastering the game of go with deep neural networks and tree search". Nature, 529(7587) pp. 484-489 (2016)

APPENDIX A.

In Section V-A, 14 of 29 features are explained. Here the other 15 features are briefly explained. Please note that the additional cost for calculating these 29 features for each Black move is not so expensive in fact, because almost all feature values can be calculated within the procedure that the program decides the White move.

- **trstdbefore, trstdafter, trstddiff**, standard deviation of territory advantages, before/after the move and its difference. They are calculated with **trbefore, trafter, trdiff**, and representing how unclear the game result is.
- **dist01, dist02, dist21, dist0b, dist2b**, Euclidean distances between two of { the last White move (0), the next White move (2), the actual Black move (1), and the estimated best move (b) }.
- **own2before, own2after, own2diff**, averaged ownership of Black stones on 3×3 area neighboring the Black move. Values before/after the move, and its difference.
- **bdecav, wdecav**, average ownership decreasements of all Black/White stones, by the next White move. When **bdecav** is high, it means Black stones are weaken by the next White move, because of losing chance to defend.
- **bdec30, wdec30**, the number of Black/White stones which their ownerships are decreased by 0.3, by the next White move.

In this paper, 7 of them, **own2before, own2after, own2diff, dist0b, dist2b, bdecav, bdecav** were removed after a test described in Section V-D and not used in the last evaluation. However, if more learning set and/or stronger program can be used, it may be better to use these 7 and more features.

APPENDIX B.

Figure 3 shows the 1st to 54th moves of game-3, evaluated in Section V-E. In fact, the Black player resigned after White 77, but only 54 moves are shown for readability. The bad moves detected by our method, and comments by the professional are as follows:

- 8th, group-1 (bad shape). OK.
- 14th, group-1. OK.
- 18th, group-1. OK.
- 24th. This move is not good, but not detected.
- 30th, group-3 (far from hot area). This move is not so bad and another type would be better.
- 32nd, group-3. OK.
- 38th, group-1. This move is not so bad.
- 46th, group-1. Another type will be better.
- 54th. This move is fairly bad, but not detected.
- anyway, detection and labeling are at a useful level.

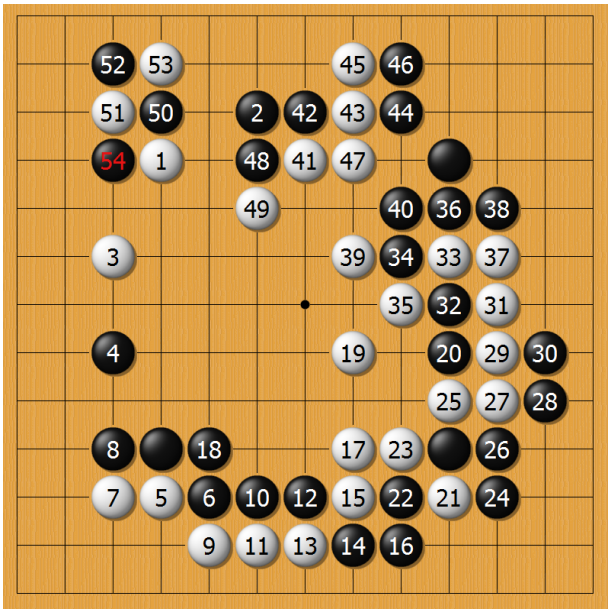

Fig. 3. Game-3 labeled by our method, up to the 54th move.