An Automatic Tuning of Game Playing System based on the Realization-Probability Search

Jun Nagashima[1], Tsuyoshi Hashimoto[3], Makoto Sakuta[4], Jos W.H.M. Uiterwijk[5], and Hiroyuki Iida[1,2]

[1] Department of Computer Science, Shizuoka University
[2] PRESTO, Japan Science and Technology Agency
[3] Faculty of Engineering, Shizuoka University
[4] School of Science and Engineering, Ishinomaki Senshu University
[5] Department of Computer Science, Universiteit Maastricht

E-mail: {cs8066,hasimoto}@cs.inf.shizuoka.ac.jp,
sakuta@isenshu-u.ac.jp, uiterwijk@cs.unimaas.nl, iida@cs.inf.shizuoka.ac.jp

Abstract. Realization-Probability Search (RPS) searches deeper after probable moves and shallower after improbable moves. It can efficiently perform a selective search similar to forward pruning and selective extensions, while using a considerably simpler algorithm in a systematic way. RPS calculates move-category probabilities from master games. Even without master games, Automatic Realization-Probability Search (ARPS) enables a computer to acquire move-category probabilities from self-play games. ARPS learns reasonable search parameters automatically. This paper presents a game program automatic design scheme with a focus on the tuning of evaluation function within the ARPS framework. It enables to make a reasonably strong program automatically for any game. Experiments performed with Lines of Action show the effectiveness of the proposed idea.

1 Introduction

Most approaches to computer game playing are based on game tree search and position evaluation. While these approaches have been very successful for many games including Chess and Othello, it has been less successful when applied to games with high branching factors and complex positions, such as Go and Shogi. One would also argue that it is still difficult to make a strong program for a new game since making a strong program needs a lot of domain-specific knowledge. Therefore, it is currently an important challenge to create a strong program possibly with little domain knowledge and less hand-tuning task.

In this paper we propose a method to automatically generate evaluation components to be used for search control. Using the method, programmers can tune automatically both an evaluation function and search control. The method is applicable to any game because it does not assume any domain specific knowledge. It also enables to make a stronger program with less cost than before. Section 2 describes some methods that have been used for tuning an evaluation function and search control. Section 3 shows a method we propose. In Section 4 we apply the proposed method to our Lines of Action program (T-T). Section 5 presents the results of self-play experiments and show that the proposed method improves the strength of our program. Finally, conclusions are given as well as future works.

2 Related Work

Many ideas to enhance search control and tune an evaluation function have been proposed. We mention some of them related to our present work. We first describe the basic idea of Realization-Probability Search. We then explain the extension of Realization-Probability Search, called Automatic Realization-Probability Search. We also give a short sketch of the Temporal Difference Learning.

2.1 Realization-Probability Search

In the Realization-Probability Search (RPS) [9], at first moves are classified into many categories by using several features like check, capturing piece, attack, and so on. Note that a human determines these features beforehand. Then the statistics of all move categories are obtained from master games. Move-category probability reflects how often masters play such a move.

The probability of a move category is calculated as follows.

$$M_p = \frac{N_p}{N_e}$$

M_p represents the probability of a move-category. N_p means the frequency of the chosen move's category and N_e the number of times this category is applicable.

The realization probability at a position in a search is calculated as follows.

$$RP_{n+1} = RP_n x M_p$$

RP_n means the realization probability at the parent position.

RPS performs iterative deepening while decreasing an upper bound on the realization probability instead of increasing the search depth. Since a "probable" move has a large probability while an "improbable" one has a small probability, the search proceeds deeper after probable moves and shallower after improbable moves. This means that RPS can efficiently perform a selective search similar to forward pruning and search extensions, while using a considerably simpler algorithm in a systematic way. The fact that GEKISASHI won the 12th World Computer Shogi Championship is an indication of the superiority of RPS in Shogi.

2.2 Automatic Realization-Probability Search

In many games such as LOA we face the problem of having much fewer good game scores available for calculating the statistics of the move categories than in Shogi. Then Automatic Realization-Probability Search (ARPS) was proposed[2]. It can obtain the move-category probabilities through automatic learning, which makes RPS in principle applicable to any game.

In ARPS, at first moves are classified into many categories by using several features chosen by a human. The initial value of every move-category probability is set arbitrarily to a fixed value between zero and one (a value of 0.5 was chosen). Then many automatic self-play games are played using RPS. At every turn the number of possible moves and the chosen move-category at the root are recorded. After a certain number of games are played, each move-category probability is recalculated. Then the self-play is repeated using the new probabilities until no significant changes in the probabilities occur.

Since ARPS calculates the move-category probabilities through automatic learning, it works well in a game that has a good evaluation function. An evaluation function is important in ARPS to obtain reasonable probabilities. To summarize, ARPS can provide a strong self-learning mechanical brain for games only when a good evaluation function and move categorization is available.

2.3 Temporal Difference Learning

Many AI systems use evaluation functions for guiding search tasks. In the context of strategy games they usually map game positions into the real numbers for estimating the winning chance for the player to move. Evaluation function construction is always a hard problem, even when focusing only on the particular domains.

There are two important issues: selecting features and combining them. In the 1950s Samuel proposed a way for automatically tuning weight [6] similar to Temporal Difference Learning or TD-Learing in short [7].

TD-Learning learns with the predict of the near future. A computer can tune factors (features) automatically through self-play games because TD-Learning does not require teaching signals. Weights between factors are updated after every move as follows.

$$W_{t+1} = W_t + \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_W P_k$$

W is a vector of weights of evaluation element. P is prediction. P is the wining rate of the position. α affects the rate of learning. λ is the weight for P. $\nabla_W P_k$ is the vector of partial derivatives with respect to each component of W.

TD-Gammon [8], a master-level backgammon program is an example of learning evaluation function using the TD-Learning.

3 Automation of tuning

To make a strong game playing system, parameters of search control and evaluation function have to be tuned properly. Applying ARPS after learning evaluation function by TD-Learning enable to a make strong program automatically. To do this, there are following four things to deal with.

1. selecting factors for evaluation function
2. tuning factors for evaluation function
3. classifying moves into some categories
4. calculating move-category probability

ARPS is able to deal with the task (4) automatically. TD-Learning is also able to deal with the task (2) automatically. This paper shows a method that can deal with the task (3) automatically. Now only you have to do is selecting features.

We classify according to whether a move increases evaluation value of each evaluation element, or it reduces to classify moves into some

Table 1. Classifying moves.

	$f(a') - f(a)$	$g(b') - g(b)$	$v' - v$
Category 1	+	+	+
Category 2	+	−	+
Category 3	+	−	−
Category 4	−	+	+
Category 5	−	+	−
Category 6	−	−	−

categories. The following situations are considered as an example. There are two evaluation factors: A and B. The evaluation value of position, v, is given by the following formula.

$$v = f(a) + g(b)$$

$f(a)$ is an evaluation value of factor A, and $g(b)$ is an evaluation value of factor B. After playing move X, the evaluation value of position changes as v'. The evaluation value of factor A also changes $f(a')$ and the evaluation value of factor B changes $g(b')$. Using $f(a)$, $g(b)$, v, $f(a')$, $g(b')$ and v', we can classify moves as shown in Table 1. If $f(a) = 10$, $g(b) = 70$, $f(a') = 30$ and $g(b') = 40$, move X is classified into Category 3.

The moves that evaluation falls directly are seldom played when search depth is sallow. When search depth becomes deeper, such kind of moves could be played because those moves could lead an advantageous position. However, we expect that the moves that evaluation raises directly be more often played than the moves that evaluation falls directly. If all move-category has very high probability or very low probability, RPS can search deeper in "probable" position and sallower in "improbable" position. Our method uses evaluation factors that mean directly advantage or disadvantage; therefore we expects all move-category has very high probability or very low probability.

4 Application to Lines of Action

We implement our proposed method to Lines of Action program (T-T) to examine its effectiveness. There are two reasons why we apply the method to LOA. One is that we already implement ARPS to (T-T) and ARPS works well on (T-T). The other one is that LOA is not so complex. According to [12], the average branching factor of LOA is about 25 and average game length is 40; therefore we expect that TD-Learning works well.

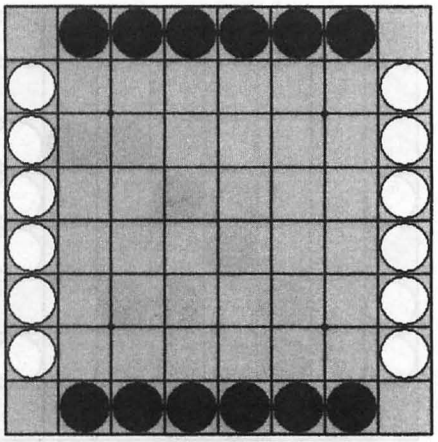

Figure 1. The initial position.

We give a short description of the rules of LOA. Then we show the detail of our method applied to LOA.

4.1 The rules of LOA

LOA is two persons zero-sum finite certain perfect information game with 8 x 8 board. The initial position is given in Figure 1. If stones are side by side, they are connecting. The goal of this game is to connect all of player's stones. Figure 2 is an example of the terminal position. In this figure all of black's stones are connected; therefore this game is won by Black.

The line has an important meaning, as it is also the name of this game. Each player must move its stone. A move takes place in a straight line, exactly as many squares as there are stones anywhere along the line of movement. A player can jump over its stones but cannot jump over opponent's stones. A player can capture opponent's stone by landing on them. Figure 3 is an example.

The stone, placing in fourth line from top and fourth line from left, can move to six squares arrowed in the figure. That stone cannot move left side because there is own stone in the destination square. That stone also cannot move down side because that stones cannot jump Black stone. A move to down right side is a capture move. If you want to know more detail, please refer to Home Page of Mark Winands [11].

4.2 Implementation in LOA program (T-T)

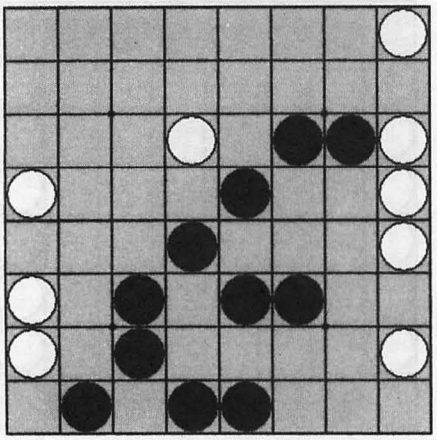

Figure 2. An example of terminal position.

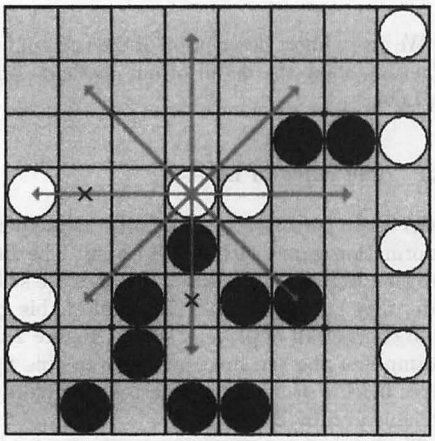

Figure 3. An example of legal moves.

Evaluation function learning There are four evaluation factors used in evaluation function of (T-T). Referring evaluation factors used in YL and MIA([1][13]), most strong LOA programs, we select four factors as follows.

- Relative position evaluation from the center of gravity
- Line pattern evaluation
- 3x3 pattern evaluation
- Player to move

As relative position evaluation from the center of gravity, we give values every 0.5 distance. This factor means concentration of stones. If stones are concentrate, each stone can easily connect; therefore if stones are concentrate, a player can easily win. The factor of concentration is implemented

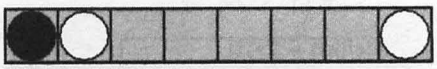

Figure 4. An example of Line pattern 1.

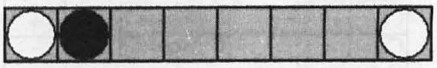

Figure 5. An example of Line pattern 2.

in most LOA programs although the way to implement may differ.

Line pattern evaluation is the factor that evaluates which player has the advantage in a line. For example, White has more advantage in Figure 4 than in Figure 5 because White can connect easier in Figure 4 than in Figure 5. (T-T)'s evaluator evaluates line pattern evaluation if a line is longer than 5 squares; therefore there are 2382 factors in Line pattern evaluation.

3x3 pattern evaluation is the factor that evaluates which player has the advantage in the 3x3 area around stone. For example, White has more advantage in Figure 6 than in Figure 7. (T-T)'s evaluator evaluates 651 3x3-patterns. Line pattern evaluation and 3x3-pattern evaluation are implemented to evaluate blocking opponent's stones and strength of connection.

The factors of blocking and connectedness are evaluated in YL and MIA evaluator although the way to implement may differ. Player to move is implemented in MIA evaluator. This factor means an advantage of moving side. (T-T)'s evaluator calculates the sum of those four factors as an evaluation value.

These four factors are well tuned by TD-Learning. The prediction P is set as follows.

$$P = sigm(\frac{v}{3000})$$

v means evaluation value. ¿From the formula for updating weights, the amount of updating $\Delta w_{t,i}$ is given by the following formula.

$$\Delta w_{t,i} = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \frac{\lambda^{t-k} P_k(1 - P_k) x_{k,i}}{3000}$$

$x_{k,i}$ is an input of evaluation i^{th} factor. We initialize all weights by giving randomized values distributed from -10 to 10. We decrease α from 100000 to 1. λ is set as 0.8 and the threshold search depth is set as 4. The factors are learned through 20000 self-play games. The factors of relative position evaluation from the center of gravity are learned as shown in Table 2 and Figure 8.

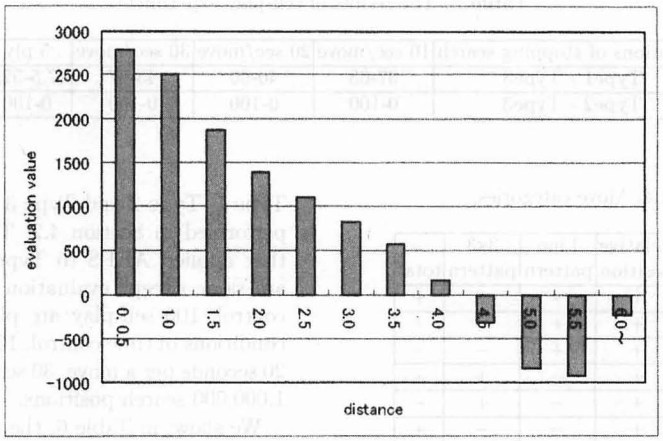

Figure 8. Results of factor's weight learning.

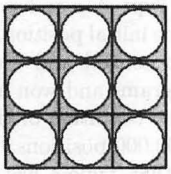

Figure 6. An example of 3x3 pattern 1.

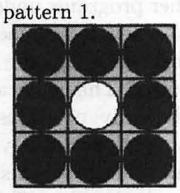

Figure 7. An example of 3x3 pattern 2.

Self-play experiments are performed to examine the accuracy of this evaluation function. We prepare three types of programs.

- Type1, (T-T) 2002 version
- Type2, (T-T) 2003 (before learning) version
- Type3, (T-T) 2003 (after learning) version

Type 1 is the version that took part in 7th Computer Olympiad held on August 2002 and won the bronze medal. This version evaluates only the factor of relative position evaluation from the center of gravity. The weights of factors are tuned by hand. Type 2 is the version before learning evaluation function and Type 3 is the version after learning. These three programs are the same except evaluation functions. Their search algorithm is brute force.

Table 2. Results of factor's weight learning.

Distance from the center of gravity	Weight	Distance from the center of gravity	Weigh
0, 0.5	2793.86	3.5	578.87
1.0	2508.31	4.0	159.76
1.5	1868.31	4.5	-317.57
2.0	1387.73	5.0	-847.50
2.5	1092.33	5.5	-927.62
3.0	832.42	6.0 ~	-233.42

100 self-play games are performed under the five different conditions on time control: 10 seconds per a move, 20 seconds per a move, 30 seconds per a move, 5 ply and 6 ply.

We show, in Table 3, the experimental results. We deal with a draw game as 0.5 win and 0.5 lose. Type 3 won under all conditions of time control against Type 1 and Type 2; therefore Type 3 could learn the best evaluation function.

Search control learning Four factors are selected as evaluation factors. Using these factors, we apply ARPS. Unfortunately, we cannot use the factor of "player to move" because moving side changes after moving any kinds of moves; therefore all moves reduce evaluation value of player to move. Using another three factors, we classify moves into groups shown in Table 4.

Using the evaluation function learned in the previous experiment, we apply ARPS with the move categories. We initialize all move-category

Table 3. The results of self-play experiment.

conditions of stopping search	10 sec/move	20 sec/move	30 sec/move	5 ply	6 ply
Type1 - Type3	37-63	40-60	43-57	47.5-52.5	32-68
Type2 - Type3	0-100	0-100	0-100	0-100	0-100

Table 4. Move categories.

	Relative position	Line pattern	3x3 pattern	total
Category 1	+	+	+	+
Category 2	+	+	−	+
Category 3	+	+	−	−
Category 4	+	−	+	+
Category 5	+	−	+	−
Category 6	+	−	−	+
Category 7	+	−	−	−
Category 8	−	+	+	+
Category 9	−	+	+	−
Category 10	−	+	−	+
Category 11	−	+	−	−
Category 12	−	−	+	+
Category 13	−	−	+	−
Category 14	−	−	−	−

Table 5. Results on the learned move-category probabilities.

Category	probability	Category	probability
Category 1	41.28%	Category 8	4.87%
Category 2	18.91%	Category 9	0.98%
Category 3	3.86%	Category 10	4.29%
Category 4	21.04%	Category 11	3.68%
Category 5	4.97%	Category 12	4.96%
Category 6	8.22%	Category 13	2.81%
Category 7	6.11%	Category 14	1.62%

probability as 50% before applying ARPS. Through 500 self-play games under the 5-ply search, we calculate move-category probability. The move-category probabilities given by ARPS is shown in Table 5 and Figure 9.

5 Experiment

Self-play experiments are performed to evaluate our method. We prepare four different programs for the experiments.

- Type 1, (T-T) 2002 version
- Type 2, (T-T) 2003 (before learning) version
- Type 3, (T-T) 2003 (after learning) version
- Type 4, (T-T) 2003 (after apply ARPS) version

Type 1, Type 2 and Type 3 are used experiment performed in Section 4.2. Type 4 is the version that applied ARPS to Type 3. These programs are same except evaluation function and search control. 100 self-play are played under the four conditions of time control: 10 seconds per a move, 20 seconds per a move, 30 seconds per a move and 1,000,000 search positions.

We show, in Table 6, the results of the experiments. Since type 4 is applied ARPS, it took an extra cost for classifying moves. The search speed of Type 1 is 100,000 positions per second, Both Type 2 and Type 3 search 58,000 positions per second, while Type 4 searches 36,000 positions per second in the initial position. Type 4 has more complex search control and evaluation function than other programs and won against other programs under the condition of time control when it searches 1,000,000 positions.

This means that Type 4 has good search control and accurate evaluation function. Type 4 also won against other programs under the conditions of time control when it searches certain times. Type 4 only lost against Type 3 under the 10 second time control. This is because Type 4 cannot search sufficiently positions within 10 second. From this result, we observe that Type 4 is stronger than all other programs. In other words, the proposed method makes (T-T) stronger.

6 Conclusion and Future Work

In this paper we proposed a method that enables to automatically generate evaluation features to be used in search control in Automatic Realization-Probability Search. The proposed method enables a programmer to automatically tune both evaluation function and search control We showed an application in the domain of LOA using our program (T-T). Experiments performed showed the effectiveness of the proposed idea.

We will apply the method to other more complicated games such as Azazons or Shogi. The method needs to select evaluation factors. Kaneko [3] and other researchers have investigated how to learn evaluation factors automatically. Following their ideas, the task for creating a strong game program will be completely automatic.

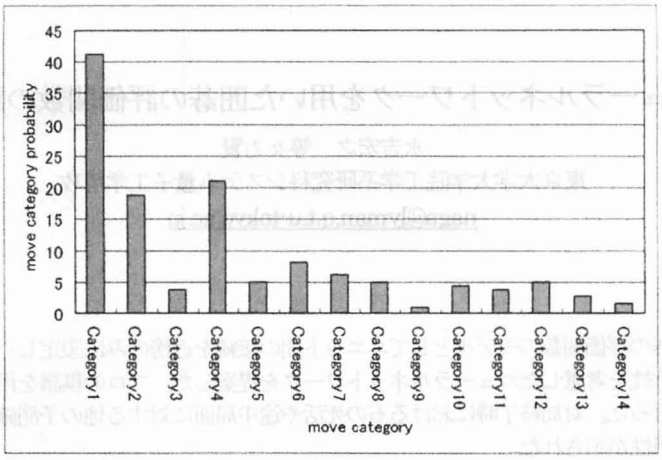

Figure 9. Results on the learned move-category probabilities.

Table 6. The results of experiments.

time control	10 sec/move	20 sec/move	30 sec/move	1,000,000 positions
Type1 - Type4	30-70	32-68	34-66	31.5-68.5
Type2 - Type4	0-100	0-100	0-100	0-100
Type3 - Type4	55-45	42-58	38-62	26-74

References

1. Billings, D. and Björnsson, Y.: SEARCH AND KNOWLEDGE IN LINES OF ACTION, accepted for publication, ACG10 (2003)
2. Hashimoto, T., Nagashima, J., Sakuta, M., Uiterwijk, J.W.H.M. and Iida,H.: Application of Realization Probability Search for Any Games - a case study using Lines of Action-", Game Programing Workshop 2002
3. Kaneko, T. and Yamaguchi. K.: Pattern Selection Problem for Automatically Generating Evaluation Functions for General Game Player, The 7th Game Programming Workshop, pp.28-35 (2002)
4. Matsubara, H. and Takizawa, T.: How Shogi Programs Become Such String As Amature 4-dan, Journal of the Japanese Society for Artificial Intelligence, Vol.16, No.3, pp.379-384 (2001)
5. Sakuta, M., Nagashima, J., Hashimoto, T. and Iida, H.:
   Application of the methods developed in the endgame search of shogi to Lines of Action, Journal of the Information Processing Society of Japan, Vol.43, No.10, pp.2964-2972 (2002)
6. Samuel, A.L.: Some studies in machine learning using the game of checkers, IBM Journal of Research Development 3(3) 211–229. (1959)
7. Sutton, R. S.: Learning to predict by the methods of temporal differences, Mach. Learning 3, (1988), pp.9-44.
8. Tesauro, G.: Temporal Difference Learning TD-Gammon, Comm. ACM 38(3) 58–68. (1995)
9. Tsuruoka, Y., Yokoyama, D., Maruyama, T. and Chikayama, T.: Game-Tree Search Algorithm Based on Realization Probability, The 6th Game Programming Workshop, pp.17-24, (2001)
10. Usui, K., Suzuki, T. and Kotani, Y.: Parameter Learning Using Temporal Difference in Shogi, http://shouchan.ei.tuat.ac.jp/~shougi/old_shougi/presentation/1999-10-07P.pdf
11. Winands, M.H.M.: The Rules, Mark's LOA Homepage, http://www.cs.unimaas.nl/m.winands/loa/rules.html
12. Winands, M.H.M.: Analysis and Implementation of Lines of Action, M.Sc. Thesis, Universiteit Maastricht, The Netherlands (2000).
13. Winands, M.H.M., van den Herik, H.J. and Uiterwijk, J.W.H.M.: AN EVALUATION FUNCTION FOR LINES OF ACTION, accepted for publication, ACG10 (2003)