GAME-TREE SEARCH ALGORITHM BASED ON REALIZATION PROBABILITY

Yoshimasa Tsuruoka[1] *, Daisaku Yokoyama*[1] *and Takashi Chikayama*[1]

Tokyo, Japan

ABSTRACT

In games like chess, the node-expansion strategy significantly affects the performance of a game-playing program. In this article we propose a new game-tree search algorithm that uses the realization probabilities of nodes for deciding upon the range of the search. The realization probability of a node represents the probability that the moves leading to the node will actually be played. Our algorithm expands nodes as long as the realization probability of a node is greater than the threshold. Therefore, it spends little computational resource on unrealistic moves, resulting in a more effective search. We have implemented this algorithm in a Shogi-playing program. Experimental results show that the proposed algorithm achieves state-of-the-art performance on a standard test suite for computer Shogi. Moreover, its performance gain is equivalent to a speed-up of more than two.

1. INTRODUCTION

Shogi is a chess-like game, which is very popular in Japan. In the context of game-tree searching, the biggest difference between chess and Shogi is the reuse of pieces. In Shogi, once you have captured an opponent's piece, you can reuse the piece by dropping it on any empty square in any subsequent turn. This rule makes the branching factor of Shogi considerably larger than that of chess, especially in endgames where players usually have a large amount of pieces in hand. Today, the majority of strong Shogi programs use many forward-pruning techniques (Iida, Sakuta, and Rollason, 2002). Due to the larger branching factor of Shogi, one can hardly achieve state-of-the-art performance with the brute-force (full-width) search, which once dominated the computer-chess scene. Since those forward-pruning techniques heavily rely on knowledge-intensive heuristics, they require a huge amount of tuning effort and domain-specific knowledge.

In games like chess, the node-expansion strategy significantly affects the performance of a game-playing program. Most world-class chess programs use 'depth' as the primary criterion for determining the range of the search and incorporate many pruning and extension techniques, such as null-move forward pruning (Donninger, 1993), fail-high reductions (Feldmann, 1994), check extensions, recapture extensions and so on. Alternative search strategies are theoretically promising but the results have still to be demonstrated in practice (for example, Conspiracy Number Search (McAllester, 1988; Schaeffer, 1990; Lorenz *et al*., 1995), B* Probability based search (Berliner and McConnell, 1996)). The situation is similar in the world of computer Shogi. Almost all the top-level Shogi programs adopt this depth-based strategy.

In this article we propose a new node-expansion strategy based on the realization probabilities of nodes. While the depth-based search expands nodes to a given depth, this algorithm expands nodes as long as the realization probability of a node is estimated to be greater than the given threshold. Therefore, this algorithm spends little computational resource on unrealistic moves, resulting in more effective search.

This article is organized as follows. Section 2 presents the game-tree search algorithm based on realization probability. Section 3 describes the experimental results using a standard test suite and matches against the

[1]Tsujii Laboratory, Department of Information Science, Faculty of Science, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-0033, Japan. Email:{tsuruoka,yokoyama}@logos.t.u-tokyo.ac.jp, chikayama@klic.org.

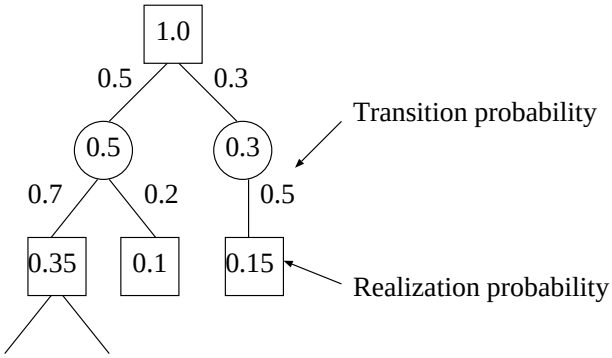

Figure 1: Realization probability and transition probability.

conventional depth-based search. Section 4 discusses the relationship between the proposed algorithm and related work. Finally, Section 5 offers some concluding remarks.

2. SEARCH ALGORITHM BASED ON REALIZATION PROBABILITY

The development of our algorithm was motivated by the observation that expert Shogi players do not determine the range of the search (i.e., node expansion) by depth. Although no one has revealed the node-expansion mechanism in the brains of expert Shogi players, there are some observations suggesting that they use a kind of 'likelihood' of a position as a criterion of whether they should further investigate the position. The experts carefully examine the positions that would be realized by a series of moves such as a sequence of re-captures. Conversely, they give little thought to the positions where important pieces have been lost without compensation.

In order to express such 'likelihood' numerically and make computers search in a similar way to human expert players, we consider the realization probabilities of positions.

2.1 Realization Probability

The realization probability of a node represents the probability that the moves leading to the node will actually be played. By definition, the realization probability of the root node is 1. Given transition probabilities, we can calculate the realization probability of a node in a recursive manner:

$$P_x = P_m \cdot P_{x'} \tag{1}$$

where P_x is the realization probability of a node x, $P_{x'}$ is the realization probability of the parent node x', and P_m is the transition probability by a move m, which changes the position x' into x. Figure 1 shows an example of realization probabilities in a search tree.

Since transition probabilities are less than 1, the probability of a node gets smaller as the search goes deeper. When the realization probability of a node becomes smaller than the predefined threshold, the node becomes a leaf. In other words, the cut-off test is done by checking whether the realization of the node is smaller than the threshold.

It should be noted that if you take the logarithm of Equation (1), you see that the calculation can be conducted by adding the logarithms of probabilities. This is done in our actual implementation. However, we use the notation of Equation (1) in the remainder of this article to emphasize that realization probabilities are calculated as a result of the multiplication of transition probabilities.

2.2 Transition Probability

The transition probability of a move is the probability that the move will actually be played. Obviously, when there is only one legal move in a position, the transition probability of the move is 100%. This means such moves do not reduce the realization probability and the lines including such moves are searched more deeply.

However, we usually have multiple legal moves in a position. In such cases, we calculate the transition probabilities of moves according to the category to which the moves belong. The categories include captures, re-captures, escape moves, promotion moves, checks, and so on. The probability of each category is calculated from actual game records played by professional Shogi players [2] . The probability of a category P_c is calculated as,

$$P_c = \frac{n_p}{n_c} \tag{2}$$

where n_c is the number of positions in which moves of the category c are legal, and n_p is the number of positions in which one of the moves of this category was actually played.

We use the probability of a category as the transition probability of a move that belongs to the category. When a move belongs to multiple categories, we use the category which has the highest probability. However, it should be noted that the probability of a category does not directly correspond to the transition probability of a move. The reason is that, if a category contains multiple legal moves at a position, the probability that one particular move in this category is played is smaller than the probability of the category. Nevertheless, the probability of the category gives an upper bound of the transition probability of a move. Currently, we have no better alternative to obtain the transition probability of a move.

Table 1 shows some representative categories and their probabilities. The category with the highest probability is 're-captures with material gain[3]', whose probability reaches as much as 89 percent. Therefore, lines including 're-captures with material gain' are more deeply searched in our algorithm, while lines including, for instance, 'capture with material loss' are searched shallowly .

2.3 Re-Search

If you simply implement the cut-off criterion based on the realization probability described above, you would face a serious problem. The problem is a kind of the horizon effect caused by the moves with a low transition probability. Because such moves can turn a node into a leaf very easily, the root node is prone to select such moves when the position is actually disadvantageous and deeper search would reveal it.

To avoid this problem, our algorithm performs a deeper search for the moves that have updated the current best value (re-search). The question here is: what transition probability value should be used? From the standpoint of transition probability, we should use the probability that the move which updates the current best value will actually be played. However, such a probability cannot be obtained from game records. Therefore we simply set the value to 0.5.

2.4 Algorithm

Except for the cut-off test and the procedure of re-search, the algorithm is identical with the normal alpha-beta search. Figure 2 shows the algorithm in C++ like code. The cut-off test checks whether the realization probability of the node is smaller than the predefined threshold. For moves with probability smaller than 0.5, the algorithm first conducts a preliminary search. Since the purpose of the preliminary search is to see whether the move will update the current best value, null-window search suffices. If the returned value indicates an update, the algorithm next conducts a re-search by which it obtains the exact value of the move. For moves with probability larger than 0.5, it conducts a normal search.

[2] We used 600 games contained in the CD-ROM "The collection of games played by Yoshiharu Habu (in Japanese)."
[3] Material gain is computed by considering re-captures only.

Category	Material gain	Transition probability
recapture	+	58%∼89% *
	0	22%
	-	5%
capture	+	16%∼42% *
	0	9%
	-	2%
check	+	43%
	0	25%
	-	4%
promote Rook	+ or 0	21%
	-	1%
promote Bishop	+ or 0	20%
	-	1%
promote Silver	+ or 0	10%
	-	2%
promote Knight	+ or 0	19%
	-	2%
promote Lance	+ or 0	10%
	-	4%
promote Pawn	+ or 0	22%
	-	5%
move an attacked piece	+ or 0	12%∼69% **
attack two pieces simultaneously	+ or 0	8%∼62% ***
attack King by Pawn	+ or 0	25%
attack King by Knight	+ or 0	20%
attack Rook by Pawn	+ or 0	23%
attack Gold by Pawn	+ or 0	11%
attack Silver by Pawn	+ or 0	11%
move Pawn	+ or 0	23%
	-	4%
move promoted Bishop	+ or 0	18%
	-	3%
move Silver	+ or 0	16%
	-	1%
move promoted Rook	+ or 0	15%
	-	1%
move King	+ or 0	5%
:	:	:

* The probability depends on the material gain.
** The probability depends on the value of the attacked piece.
*** The probability depends on the value of the attacked pieces.

Table 1: Representative move categories and their transition probabilities.

3. EVALUATION

Below we briefly evaluate our ideas on realization probability. We therefore discuss the implementation (3.1), the playing strength (3.2) and the test suite (3.3).

3.1 Implementation

The proposed algorithm has been implemented in a Shogi-playing program called GEKISASHI. We call this version GEKISASHI-R to distinguish it from the version of conventional depth-based search, which is described

```
int search(Board* board,
           double realization_probability,
           double min_realization_probability,
           int alpha, int beta, Move& best_move)
{
  // Cutoff test
  if (realization_probability < min_realization_probability)
    return leaf(board);

  // Move generation
  Move move[MAX_NUMBER_OF_MOVES];
  int number_of_moves = generate_moves(board, move);

  int best = alpha;
  for (int i = 0; i < number_of_moves; i++) {

    board->move(move[i]);

    int value;
    Move dummy;
    if (move[i].transition_probability < 0.5) {
      // Preliminary search (Null window)
      value = -search(board,
                      realization_probability * move[i].transition_probability,
                      min_realization_probability,
                      -(best+1), -best, dummy);
      if (value > best) {
        // Re-search
        value = -search(board,
                        realization_probability * 0.5,
                        min_realization_probability,
                        -beta, -best, dummy);
    } else {
      // Normal search
      value = -search(board,
                      realization_probability * move[i].transition_probability,
                      min_realization_probability,
                      -beta, -best, dummy);
    }
    board->reverse();

    if (value > best) {
      best = value;
      best_move = move[i];
      if (best >= beta)
        return best;
    }
  }
  return best;
}
```

Figure 2: The realization probability based search algorithm.

later. The program performs quiescence searches in leaf nodes, where only re-captures, captures, promotions and escapes are considered. Iterative deepening is performed using realization probabilities instead of depths. At each iteration, while the depth-based algorithm increases the threshold depth, the program decreases the threshold of realization probability by dividing it by 4. The PDS algorithm (Nagai, 1998) is used for checkmate search. The program also uses common techniques to improve search efficiency, such as transposition tables, killer heuristics, and internal iterative deepening.

3.2 Playing Strength

In order to evaluate the playing strength of the proposed algorithm, we have also implemented the conventional depth-based search algorithm on GEKISASHI. We call this version GEKISASHI-D, whose evaluation function and quiescece-search procedure are identical to those of GEKISASHI-R. As stated before, one cannot achieve state-of-the-art performance with full-width search because of the high branching factor of Shogi. Most of the top-level Shogi programs use plausible move generators to narrow the search space. Unfortunately, there is no literature which gives complete information about those heuristics. We thus constructed a plausible-move generator by hand tuning. By using the plausible-move generator, GEKISASHI-D has achieved a win percentage of 91 percent (in 200 matches) against the full-width version of GEKISASHI in our preliminary experiments. The specification of the move generator is given below.

- Frontier nodes (remaining depth = 1)

  Captures, promotions, escapes, checks and effective attacks (e.g., Double Attack) are generated.

- Pre-Frontier nodes (remaining depth = 2)

  Adding to the above, moves of approaching the opponent King and castling moves are generated.

- Remaining depth = 3

  Adding to the above, moves blocking the opponent's long distance effects are generated.

- Remaining depth = 4

  Adding to the above, all moves and attacks are generated.

We have tried several common forward-pruning and extension techniques on GEKISASHI-D, including null-move forward pruning, fail-high reductions, check extensions, recapture extensions and their combinations. In self-play experiments, the version which used null-move forward pruning, check extensions and recapture extensions performed best. Therefore we used this version for the match against GEKISASHI-R.

	wins	draws	losses	win percentage
5 secs / move	109	0	91	55%
10 secs / move	137	8	55	71%

Table 2: Win percentage of GEKISASHI-R against GEKISASHI-D (GEKISASHI-D is given 10 seconds for each move).

Table 2 shows the result of the match between GEKISASHI-R and GEKISASHI-D [4]. The match consists of 200 games from 100 unique starting positions. GEKISASHI-D is given 10 seconds for each move. Notice that win percentage reaches 71 percent when both are given the same thinking time. Even when the thinking time of GEKISASHI-R is half of GEKISASHI-D, the win percentage is still above 50 percent. The results suggest that the proposed algorithm equivalently makes the depth-based search more than two times faster.

3.3 Test Suite

Matsubara and Iida (1998) provided a standard test suite for computer Shogi programs. This suite consists of 48 (mainly middle-game and endgame) positions extracted from game records played by professional Shogi players.

Table 3 shows the results. The number of correct answers was counted regardless of whether the program understood the objective of a move. The first seven rows show the results of our programs. The rest shows the results of commercial Shogi programs as listed in Matsubara (2001). Considering the total time, GEKISASHI-R exhibits comparable (or slightly better) performance to the commercial Shogi programs.

[4] The computer used in this experiment is equipped with a Pentium III 1GHz CPU.

Program	(Setting)	Total time (secs)	Corrects
GEKISASHI-R	2^{-5} *	133	24
	2^{-7} *	346	23
	2^{-9} *	802	25
	2^{-11} *	2103	31
GEKISASHI-D	5 plies *	96	21
	6 plies *	488	21
	7 plies *	3423	23
TODAI-SHOGI 3	(master level)	1159	27
GINSEI-SHOGI 2	(level 5)	3327	22
KAKINOKI-SHOGI 5	(level 7)	1638	22
AI-SHOGI 2001	(level 5)	825	24

* Cut-off threshold.

Table 3: Performance on a test suite.

4. DISCUSSION

The experimental results show a big performance improvement over conventional depth-based search. Two possible reasons why our algorithm is effective are as follows.

- Forced lines are searched more deeply.

  In computer chess, it has long been observed that forced moves should be search more deeply. In our algorihm, when there is only one legal move at a position, the transition probability of the move is 100 percent and this move does not reduce the realization probability. Thus the lines including this move are searched more deeply. Some chess programs extend the search when there is only one legal move. Our algorithm has this extension intrinsically . In Shogi, recaptures with big material gain are mostly forced. The lines including this type of moves are also searched more deeply (see Table 1).

- Little computational resource is spent on unrealistic positions.

  Our algorithm spends little computational resource on unrealistic positions. For instance, it does so when important pieces would be lost without compensation, because the moves which realize such positions rarely exist in the game records of professional Shogi players. Although there is a chance that the algorithm overlooks a tricky move which seems unrealistic at a first glance, the merit of being able to spend more computational resources on other realistic positions is greater than the danger of an overlook.

From the implementation point of view, the proposed algorithm is very similar to the depth-based search using *fractional plies* (Levy, Broughton, and Taylor, 1989), in which the search allows non-integer depth increments for some 'active' moves, such as recaptures, captures, and checks. If you take the logarithms of the probabilities in our algorithm, the transition probabilities correspond to the fractional plies. Although there are few literature sources that offer implementation specifics of fractional plies in chess programs, it is said that many strong chess programs, for instance CRAFTY (Hyatt, 1996), use the technique. Björnsson and Marsland (2001) presented a general framework for learning the weights of fractional plies. In the context of fractional plies, our algorithm not only extends the search depth for 'active' moves but also shortens the search depth for 'inactive' moves. In Shogi, the majority of moves are 'inactive' moves. Therefore, controlling the amount of the computational resource for such moves is significantly important. The latter is not done in the framework of fractional plies.

5. CONCLUSION

In this article, we presented a new game-tree search algorithm based on the realization probability and we evaluated it on a Shogi-playing program. While the depth-based search expands nodes to a certain depth, our

algorithm expands nodes as long as the realization probability of a node is larger than the threshold. Therefore, it spends little computational resources on unrealistic moves, resulting in more effective search.

Experimental results show that the proposed algorithm achieves state-of-the-art performance on a standard test suite for computer Shogi and its performance gain is equivalent to a speed-up of more than two.

6. REFERENCES

Berliner, H. J. and McConnell, C. (1996). B* Probability Based Search. *Artificial Intelligence*, Vol. 86, pp. 97–156. ISSN 0004–3702.

Björnsson, Y. and Marsland, T. A. (2001). Learning Search Control in Adversary Games. *Advances in Computer Games 9* (eds. H. van den Herik and B. Monien), pp. 157–174. Universiteit Maastricht. ISBN 90–6216–5761.

Donninger, C. (1993). Null Move and Deep Search: Selective Search Heuristics for Obtuse Chess Programs. *ICCA Journal*, Vol. 16, No. 3, pp. 137–143.

Feldmann, R. (1994). Fail High Reductions. *Advances in Computer Chess 7* (eds. H. van den Herik, I. Herschberg, and J. Uiterwijk), pp. 111–128. Universiteit Limburg. ISBN 90 6216 1014.

Hyatt, R. (1996), Crafty - chess program. ftp://ftp.cis.uab.edu/pub/hyatt.

Iida, H., Sakuta, M., and Rollason, J. (2002). Computer Shogi. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 121–144. ISSN 0004–3702.

Levy, D., Broughton, D., and Taylor, M. (1989). The SEX algorithm in Computer Chess. *ICCA Journal*, Vol. 12, No. 1, pp. 10–21.

Lorenz, U., Rottmann, V., Feldmann, R., and Mysliwietz, P. (1995). Controlled Conspiracy-Number Search. *ICCA Journal*, Vol. 18, No. 3, pp. 135–147.

Matsubara, H. (2001). Evaluation of Computer Shogi by Next-Move Test (no.2). *IPSJ SIG Notes*, Vol. 2001, No. 28, pp. 39–46. In Japanese.

Matsubara, H. and Iida, H. (1998). Evaluation of Computer Shogi by Next-Move Test (no.1). *Advances in Computer Shogi 2*, pp. 61–111. Kyoritsu Publisher. In Japanese.

McAllester, D. A. (1988). Conspiracy Numbers for Min-Max Search. *Artificial Intelligence*, Vol. 35, pp. 287–310. ISSN 0004–3702.

Nagai, A. (1998). A new AND/OR tree search algorithm using proof number and disproof number. *Complex Games Lab Workshop*, pp. 40–45.

Schaeffer, J. (1990). Conspiracy Numbers. *Artificial Intelligence*, Vol. 43, pp. 67–84. ISSN 0004–3702.