A Randomized Game-Tree Search Algorithm for Shogi Based on Bayesian Approach

Daisaku Yokoyama[1] and Masaru Kitsuregawa[2,1]

[1] Institute of Industrial Science, The University of Tokyo
{yokoyama,kitsure}@tkl.iis.u-tokyo.ac.jp
[2] National Institute of Informatics

Abstract. We propose a new randomized game-tree search algorithm based on Bayesian Approach. It consists of two main concepts; (1) using multiple game-tree search with a randomized evaluation function as simulations, (2) treating evaluated values as probability distribution and propagating it through the game-tree using the Bayesian Approach concept. Proposed method is focusing on applying to tactical games such as Shogi, in which MCTS is not currently effective. We apply the method for Shogi using a top-level computer player application which is constructed with many domain-specific search techniques. Through large amount of self-play evaluations, we conclude our method can achieve good win ratio against an ordinary game-tree search based player when enough computing resource is available. We also precisely examine performance behaviors of the method, and depict designing directions.

Keywords: Randomized Search, Monte-Carlo Tree Search, Game-tree Search, Bayesian Approach.

1 Introduction

Randomized search algorithms have great ability to explore complicated search problems [4]. It effectively utilize parallel computing method which is inevitable with current computing environment. Game tree search problems have complicated constraints for computing and is not fully utilize such kind of parallelism. Monte-Carlo Tree Search algorithm (MCTS) [2] has been developed and is getting widely used with good performance, especially in Go playing [3]. This great advance, however, does not help to achieve good performance in Shogi, Japanese chess-like game. Computer Shogi is intensively investigated for a long time, and many search techniques are developed; selective deepening with position features[14], efficient depth-first And-Or-Tree search algorithm with proof- and disproof- numbers (DFPN) [8], and evaluation function learning algorithm using order of sibling nodes [6]. Utilizing these techniques with other chess-like game-tree search techniques such as futility pruning [7], computer Shogi player achieves comparable strength with the best human players in recent years [5]. MCTS cannot achieve such performance despite of many researches in Shogi [12]. The main reason is: (1) difficult to create effective *playouts* to simulate

D.-N. Pham and S.-B. Park (Eds.): PRICAI 2014, LNAI 8862, pp. 937–944, 2014.

reasonable plays, (2) difficult to follow one long narrow path of "correct" play. Especially (2) has been considered as a common weakness for MCTS [16]. Shogi is one of the good example problem domain. Shogi has many legal moves to play, but only few move sequences are acceptable to win. Current MCTS cannot converge the search result quickly under such situations.

We propose a new randomized game-tree search algorithm based on Bayesian Approach to deal with such difficulties which current MCTS have. It changes two main concepts in MCTS algorithm; (1) Using multiple game-tree search with randomized noise appended on evaluation value as simulations. This employs current sophisticated algorithms of game-tree search and accurate evaluation functions to search a narrow correct path to win. (2) Treating evaluated value as probability distribution and propagating it through a game-tree using the Bayesian Approach concept. Multiple randomized game-tree searches bring several tree values. We treat these values as the evaluation value with error distribution, and propagate that probability distribution through min-max tree with Bayesian concept. This employs all information of randomized simulation results with min-max tree search manner. Through intensive evaluation of self-fight using a state-of-the-art computer Shogi player, we confirm our proposed method can achieve significantly stronger performance than the original player under 30-50 times bigger computing resources. Our implementation stays in preliminary stage and have large room of parallelization or other improvements, thus the proposed method is promising.

Our contribution in this paper is; (1) propose a new randomized tree search algorithm for problems which have small number of narrow path to correct answer, (2) evaluate its effectiveness through large-scale self-fight experiments of real game player with many state-of-the-art techniques.

We describe related work in Section 2 and explain our proposed method of randomized game-tree search in Section 3. In Section 4, we evaluate the effectiveness of the proposed method on the basis of self-fight results and discuss about the direction of further improvements. Section 5 summarizes the key points and mentions future work.

2 Related Work

MCTS is especially effective when the game is hard to evaluate its board position using conventional evaluation function manner, like Go. However, several researches try to apply MCTS to games which already have reliable evaluate functions such as Amazons, Arimaa or Lines of Action, and get comparable strength to the conventional methods [11][10][16]. These research uses the evaluation function to interrupt the playout sequence and to decide the simulation result (win or lose) earlier. These researches cannot employ other conventional tree-search techniques such as futility-pruning [7], transposition table, and so on. Our proposed method uses conventional game-tree search algorithm as the simulation, therefore all of these techniques can be applied.

MCTS have another weakness when treating checkmate condition. Many games have checkmate condition in endgames. MCTS only propagate win ratio and

cannot treat such kind of decided results, therefore it cannot employ reliable information in endgames. Winands et al. proposed Monte-Carlo Tree Search Solver [15], an improvement by adding another value to propagate such reliable information in min-max tree manner. Our method can employ endgame positions information in more simple way.

There are several researches tried to apply MCTS algorithm with playout simulation for Shogi [12][13]. They try to employ several improvements such as killer-heuristics, however they cannot achieve comparable strength to conventional tree search.

3 Proposed Method

Proposed method changes two main concepts in MCTS algorithm; (1) using randomized game-tree search as simulations, (2) propagating probability distribution of evaluated value using the Bayesian Approach concept. This section explains the detail.

3.1 Randomized Tree Search as Simulation

Original MCTS algorithm uses a randomly chosen move sequence as a simulation, and its result is treated as binary value, win or lose. Our method uses randomized conventional min-max tree search as a simulation, and its evaluated value is collected to express how the board position is desirable to win. Randomized values are added to the value of evaluation function at every leaf node. Multiple evaluated values are collected at one board position, and they are treated as a probability distribution of that node.

Randomized values are calculated based on two keys, P and N. P is a number that reflects a board position which is the target for evaluation function in the min-max tree search of the simulation. It is calculated by the Zoblist Hashing [17]. N is the number of simulations done at the simulations root node (that is a leaf node of the min-max tree). N is same within one simulation tree search, thus if two board positions in one simulation are identical, the added values for the evaluation functions of the two positions are same. This helps the evaluation values to keep consistent within one min-max tree search, therefore we can employ all conventional tree search techniques such as transposition table. Introducing N helps to keep consistent among the evaluated values of leaf nodes which have same number of simulations. However we does not convince that N really contributes to make a strong computer player.

3.2 Value Propagation Based on Bayesian Approach

We apply the Bayesian Approach to propagate simulation value distribution derived from multiple randomized tree search. To ease the calculation of probability distribution of each node, our method assume the value distribution of evaluation function as the discrete distribution, and express it as several pins.

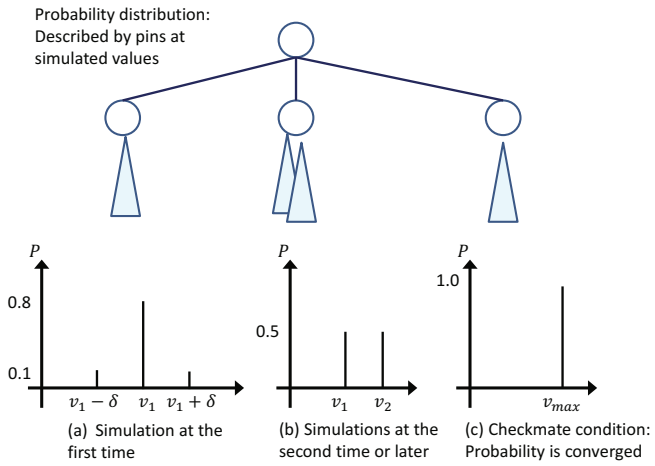

Fig. 1. Probability distributions of evaluated values constructed from simulations

Figure 1 shows the distributions of simulated values. When the first simulation search of a certain leaf node has finished with the evaluated value v_1, our algorithm assume the value distribution of the leaf node with three-pins, which has pseudo small probabilities at $v_1 \pm \delta$ (Figure 1 - (a)). If another simulation has been done with evaluated value v_2, the pseudo probabilities at $v_1 \pm \delta$ is eliminated and the probability distribution consists only from the simulation search values, v_1 and v_2 (Figure 1 - (b)). In this experiments, the evaluated value of the first simulation does not include randomized error, which means the v_1 is identical to the original evaluation function value. After the second simulation, randomized values are added to the evaluation function values. When a checkmate condition is found in a certain simulation search, the value distribution of the node is converged as a single pin to indicate the determined condition (Figure 1 - (c)).

3.3 Extension Control of Game-Tree

The algorithm of the proposed method basically execute following sequence continuously; (1) select the most important leaf node (*refine_p*) to examine in the Monte-Carlo tree, (2) if the number of simulations at the node is less than the threshold (*Simnum*), execute a simulation search from the node, (3) otherwise extend the node and grows the Monte-Carlo tree.

Parameter *Playdepth* is a threshold that defines basic strength of the player. Our proposed method will return the PV (Principal Variation) which length is longer than or equal to the $Playdepth + PVth$, where $PVth$ denotes the additional length of PV. Proposed method have probability distribution at every nodes in the min-max tree, thus we assume that each players best play is defined according to the expected value of the distribution of child node.

We implement three strategies to control the node extension in Monte-Carlo tree; Breadth First, UCB, and QSS.

- Breadth First extends the Monte-Carlo tree in breadth first manner. It limits the number of children at each node as 3, to achieve the practical execution time.
- UCB [9] is used to balance two policies, exploitation and exploration. We choose the target node to extend by recursively selecting the child node which maximize the UCB. We limit the number of children gradually reduced when the depth becomes large.
- QSS is a strategy adopted in original Bayesian Approach [1]. We select the extension target which maximize the QSS_L, which considered as the most influential node to change the value distribution of the root node. If the top 10% of leaf nodes are already reached to the depth *Playdepth*, then we extend the leaf of current PV, to reduce the computational cost.
  In selecting the extension target node, we also use the U_{all}, which is originally proposed in [1] to decide the termination condition. U_{all} denotes the difference between expected values of the root and of the best child of the root. When the U_{all} does not change for a long time, we assume that the current extension strategy does not works efficiently, and we select the leaf of PV to extend.

There are several ad-hook policies to control branching factor, extension target, or termination condition, as explained before. These methods aim to reduce execution time and achieve a practical computer player. However, we did not intensively examined about these methods.

We examine the effectiveness of these strategies in Section 4.

4 Evaluation

4.1 Experimental Environments

We implement the proposed method using a state-of-the-art computer Shogi player Gekisashi[1], and evaluate its character and effectiveness of the method. The original player program employs many domain specific techniques, and is one of the strongest player in the world.

We make many fight between original player and proposed method to evaluate the strength. We prepare a test set consists of 500 initial board positions randomly selected from the 31th move position of human play records. We make two self-fight games from an initial positon, by changing initial side of the target programs. Thus we calculate a win ratio from 1000 fights at every evaluation condition. Original player is configured to search to depth 12. *Playdepth* in proposed methods are also configured to 12. Each 1000 fights requires about 500 to 2,000 CPU hours. 95% confidence intervals are depicted at all results in the following figures.

[1] `http://www.logos.ic.i.u-tokyo.ac.jp/~gekisashi`

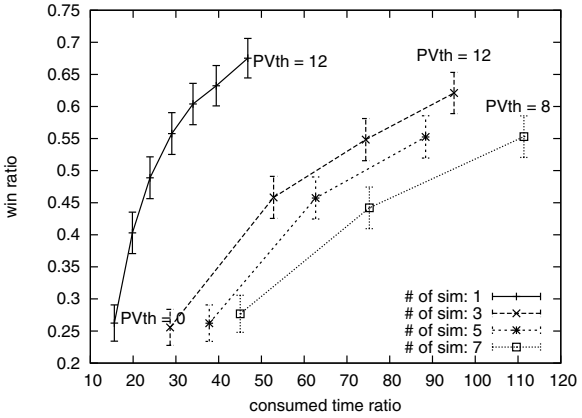

Fig. 2. Performance comparison of win ratio and consumed time under typical settings

We intensively examined the basic behaviors of our algorithm, such as the effects of simulation size, the distribution of random numbers, or pseudo distribution of initial simulation. We confirmed the algorithm can achieve comparable strength even when it uses small simulation search tree. We omit the results in this paper due to the space limitation. In the following experiments, we basically use $simdepth = 800$, $\sigma = 200$ and $\delta = 100$, where σ indicats the standard deviation of pseudo random values, and δ defines the distribution of the first simulation of a certain node.

4.2 Relationship between win Ratio and Consumed Time

Figure 2 shows the relationship between win ratio and consumed time under several typical settings using QSS extension strategy. Each series shows the result with different *Simnum* setting from 1 to 7. Each point indicates the result of different $PVth$. Horizontal axis indicates the consumed time ratio normalized by the time of original player.

The win ratio is getting better when $PVth$ becomes large at all settings. We conclude the proposed method is promising to construct a strong player. However, the win ratio seems to be slightly decreased when *Simnum* becomes large under the same $PVth$. This means that randomized search does not work effectively with QSS extension strategy.

4.3 Effects of Extension Policy

Figure 3 (left) shows the win ratio using three extension strategies. The UCB strategy requires longer execution time than the QSS with the same $PVth$ settings, thus we depicts both results of $PVth = 0$ and 4 using UCB strategy. The Breadth First strategy (denoted as "BF" series) requires far more execution time

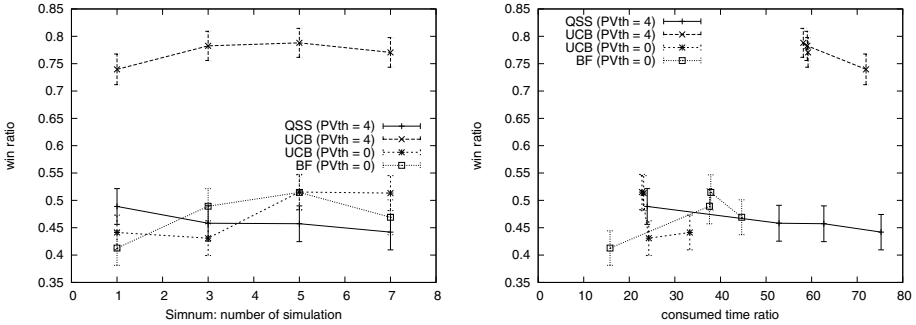

Fig. 3. Effects of extension policy compared by the number of simulations (left), and consumed time (right)

than QSS and UCB, therefore we omit to show the results with $PVth = 4$. The horizontal axis indicates *Simnum*.

When the number of simulation becomes large, the win ratio seems getting better with UCB strategy and $PVth = 0$. This means that changing strategy of tree extension seems to be effective to improve the randomized part of the proposed method. The Breadth First strategy also seems to be effective under $Simnum \leq 5$ condition, however win ratio decreasing is occured at $Simnum = 7$.

Figure 3 (right) shows the same result which depicts the consumed time ratio as the horizontal axis. Consumed time is reduced in UCB strategy when *Simnum* becomes large. It is common for the computer player that the loser side tend to consume longer time than the winner side, therefore win ratio improvement may be one of the reason of that result. UCB strategy realizes good performance with acceptable consumed time.

5 Conclusion

We propose a new randomized game-tree search algorithm based on Bayesian Approach, which uses multiple game-tree search with a randomized evaluation function as simulations and treats evaluated values as probability distribution. Through large amount of self-play evaluations, we conclude our method can achieve good win ratio against an ordinary game-tree search based player when 30-50 times larger computing resource is available. Our current implementation is preliminary and has many rooms to improve performance.

Currently, randomization does not effectively work with QSS-based extension strategy, though it works well with UCB-based strategy. We try to construct a better strategy in our future work.

Proposed method have a good characteristic to apply large-scale distributed computing. We also try to establish a distributed algorithm and evaluate its effectiveness.

Acknowledgments. This work is partially supported by JSPS KAKENHI Grant Number 26280130.

References

1. Baum, E.B., Smith, W.D.: A bayesian approach to relevance in game playing. Artificial Intelligence 97(1-2), 195–242 (1997)
2. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
3. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with Patterns in Monte-Carlo Go. Tech. Rep. RR-6062, INRIA (2006), http://hal.inria.fr/inria-00117266
4. Hart, J.P., Shogan, A.W.: Semi-greedy heuristics: An empirical study. Operations Research Letters 6(3), 107–114 (1987)
5. Hoki, K., Kaneko, T., Yokoyama, D., Obata, T., Yamashita, H., Tsuruoka, Y., Ito, T.: A system-design outline of the distributed-shogi-system akara 2010. In: 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 466–471 (July 2013)
6. Hoki, K., Kaneko, T.: The global landscape of objective functions for the optimization of shogi piece values with a game-tree search. In: van den Herik, H.J., Plaat, A. (eds.) ACG 2011. LNCS, vol. 7168, pp. 184–195. Springer, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-31866-5_16
7. Hoki, K., Muramatsu, M.: Efficiency of three forward-pruning techniques in shogi: Futility pruning, null-move pruning, and late move reduction (LMR). Entertainment Computing 3(3), 51–57 (2012), http://www.sciencedirect.com/science/article/pii/S1875952111000450
8. Kishimoto, A., Winands, M., Müller, M., Saito, J.T.: Game-tree search using proof numbers: The first twenty years. ICGA Journal 35(3), 131–156 (2012)
9. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11871842_29
10. Kozelek, T.: Methods of mcts and the game arimaa. Charles University, Prague, Faculty of Mathematics and Physics (2009)
11. Lorentz, R.J.: Amazons discover monte-carlo. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 13–24. Springer, Heidelberg (2008)
12. Sato, Y., Takahashi, D.: A shogi program based on monte-carlo tree search. In: The 13th Game Programming Workshop (2008) (in Japanese)
13. Takeuchi, S., Kaneko, T., Yamaguchi, K.: Evaluation function based monte carlo tree search in shogi. In: The 15th Game Programming Workshop, pp. 86–89 (2010) (in Japanese)
14. Tsuruoka, Y., Yokoyama, D., Chikayama, T.: Game-tree search algorithm based on realization probability. ICGA Journal 25(3), 145–152 (2002)
15. Winands, M.H.M., Björnsson, Y., Saito, J.-T.: Monte-carlo tree search solver. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 25–36. Springer, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-87608-3_3
16. Winands, M.H.M., Björnsson, Y.: Evaluation function based monte-carlo LOA. In: van den Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 33–44. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-12993-3_4
17. Zobrist, A.L.: A new hashing method with application for game playing. ICCA Journal 13(2), 69–73 (1990)