International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 12 (2019), pp. 2769-2774
© International Research Publication House. http://www.irphouse.com

Dynamic UCB Adaptation Strategy for MCTS based GVGAI

Jun Hwan Kang, Hang Joon Kim

School of Computer Science and Engineering,
Kyungpook National University, Daegu, Republic of Korea.

ORCID: 0000-0001-6592-7585 .

Abstract

Recent with amazing performance improvement of computer hardware and the generalization of AI technologies, such as deep learning and big data, game AI gained the attention of many people, and has also shown remarkable achievements in problems such as Go game, that were considered difficult to solve in the past. Among the many new AI studies, Monte Carlo Tree Search (MCTS) played the biggest role, and it is thought that game AI can solve almost any problem if given ample learning opportunities.

In this study, we propose ways to expand and improve the approach currently used in the general video game AI research aimed at solving various video games that are frequently encountered by ordinary people in real time without pre-learning. This approach suggested that, within the framework of adding a progressive history to MCTS, which is the latest method currently used, the parameters that most affect MCTS' problem pool characteristics can be dynamically converted as the game progresses, thereby demonstrating progress over the existing static method.

The specific optimal values of the reset cycle, the pre-emption period, and the delay interval to determine this dynamic change will depend on the type of game to be applied or its specific characteristics, so further studies such as how to confirm it in the future are thought to be necessary, but it is certain that performance can be improved relative to the static method just by using the appropriate values determined through the experiment.

Keywords: UCB, MCTS, Game AI, General Video Game

I. INTRODUCTION

With the recent emergence of new software-driven products and services along with improved performance of computer hardware, many new software technologies in various fields have come into the spotlight. And artificial intelligence is one of them, AI was stagnant due to the limitations of insufficient computing performance and algorithms, but now it becomes a key technology in many areas such as image recognition and language processing as the emergence of neural networks and machine learning. It was 1996 when AI first demonstrated its power in chess games, and after 20 years AI gets much stronger and can defeat human in far more complex game such as Go game. This time, AI research has started to receive real public attention whether AI really surpasses human intelligence.

In the case of Go game, once there was a question whether AI can find a workable algorithm, for it is very difficult to access all cases of the Go game as efficiently as in chess games. Monte Carlo Tree Search (MCTS) algorithms [1] [2] open a new door to solve that difficulty, and demonstrated that it is possible to solve problems which requires extensive number search effectively, through large-scale arbitrary enforcement. MCTS algorithms are very effective algorithms in board games such as Go, where game situation change is static and all the opponent's information is open so you don't have to deduce the opponent's hidden information. Recently most Go AI systems used MCTS-based algorithms and shows the most visible performances. AI's great success comes from applying machine learning to each part of MCTS algorithm in a sophisticated way. In addition, research on Reinforcement Learning by applying this approach to game, it showed that simple joystick-based one-man games can be solved completely solved through machine learning and MCTS algorithm.

However, there are clear limitations to incorporating these studies into other real-life commercial games. Many video games require immediate human response and understanding of complex rules, and it is often not a static game like a Go game, that of limiting actionable units to perform numerical searches in turn. In dynamic situations, action is required in real time over time, given information about the opponent is limited, learning opportunities is limited, and immediate status changes occur. Traditional AI methods are difficult to apply and it requires a different approach. Usually we need an AI agent who can make an immediate real-time response are required according to the dynamic change of state. It should make an accurate inference of the opponent's condition and predict enemy behavior based on it within a limited time.

In this paper, we studied general game playing research which tries to find a single algorithm that can be applied to the various forms of one-time game environment, and proposes to improve its performance by adapting its parameters to optimize its behavior in real time according to the changing situation, so it can actively respond to different environments. This paper follows the research environment provided by GVGAI, a general video game play artificial intelligence conference, and aims to achieve overall performance improvements by applying the same artificial intelligence under various conditions

II. BACKGROUND

A. General Video Game Artificial Intelligence

General Game Playing (GGP) is a study that is designed to overcome limitations where AI algorithms of current games are mostly designed for single environments. The goal is to design a single game AI that can work well in as many different environments as possible, and furthermore, based on it, we want to achieve better results than the traditional method of applying it to real games.

The GGP is a field of research that began in video game scripts. Research was conducted in two main ways. One method is to use computer vision to obtain video recognition data from game monitor graphics and design AI to understand them, and the other is to design game AI by referring to data needed for AI provided inside the game. [3] [4] For Arcade Learning Environment (ALE), a game AI research platform using machine learning, related research [5] has already been carried out, and it has now successfully reached nearly 100%. This raised the need for AI research that showed good performance without relying on machine learning. The General Video Game AI (GVGAI), a general video game for GGP research presented under these conditions, is designed to operate for the first time without any chance of pre-learning for all games[6][7]. This is to develop AI, which is close to human intelligence, even when there is no help in learning machines, by presenting the same situations that humans have done for the first time. In the case of GVGAI, it was designed by referring to internal data, which excludes the gap in image recognition efficiency due to differences in performance of computing equipment, the burden of establishing video recognition environment, and the risk of error generation, and the advantage of conducting research focused on AI algorithms using only internal game data. Accordingly, research was conducted by designing and operating AI by accommodating research platforms provided by GVGAI. The GVGAI competition is an international competition that is co-hosted by University of Essex and DeepMind and held periodically at IEEE WCCI and IEEE CIG.

A total of 82 test games are available in the basic test environment for research, and are operated by attaching AI developed according to the interface of a given JAVA platform. The calculation time allocated for each AI is 40ms, and calculations and actions must be performed and completed within a given frame time. The actionable unit is presented as an action per ton, such as Go game, but AI requires a dynamic response capability because the time limit given to a one-turn is limited to a real-time gaming environment, which is limited to 30 frames per second. If the action cannot be determined in time because the calculation has not been completed, then the action is taken without taking any action. The 82 tests given are for pre-research purposes, and since other randomly prepared gaming environments are applied at the main competition to measure AI performance, they are designed to eliminate efforts to increase the odds by increasing the odds or performing machine learning in advance.

The progression of the game follows the environment in which people play themselves. Based on the interface that you type and play on the player's keyboard, most of the case, you can only do up to five different actions each time, including up and down keys and special action keys. Most of the objects that exist in the game also behave regularly, and even if they move randomly, the total number of possible actions is limited. The movement of the main character in the game is also limited within five actions, si its search space is much limited then that of the chess and Go game, but still it’s large enough to search easily. However, the GGP does not necessarily have to perform a full search, as it is aimed at exploring the number of cases in time and not at finding the best number, but at satisfying the final win-win conditions by repeating the best actions available now by exploring the number of cases in which they are seen. That is, it is not a problem that requires an algorithm to fully resolve its complexity at once, and rather than obtain the highest global maximum at once, the goal is to somehow reach the level of value that satisfies the winning condition by constantly exploring the global maximum value. The finals have a total of three categories: one-man play, two-man competition play and machine learning-based game play. For machine learning-based competitions, we excluded it because it deviates from the research topic of this paper. For two-player competition, it is possible that the AI type in a given gaming environment may significantly change the paid. This was considered to be inappropriate for measuring the objective performance of AI, such as [8] when the developed AI is valid or vulnerable to a particular style of AI design.

B. Upper Confidence Bound

Algorithms mainly used by AI agents with excellent GVGAI are genetic algorithms, avarice algorithms, and Monte Carlo Tree Search. They include MCTS algorithms. Although many genetic and rule-based algorithms were early in the competition, most of the MCTS algorithms are currently at the top of the list. The Monte Carlo Tree Search algorithm is the search algorithm of the tree structure to find the best desired goal among the number of cases. Like conventional game tree searches, there are features that are constantly exploring and expanding to find as many cases as possible and choose the most suitable one. However, since conventional tree-search algorithms follow navigation rules based on existing data structure formats that recursively search by fixing the sequence of searches with depth-first or width-first searches, there is a fatal problem where the proper global maximum value cannot be obtained until all searches are complete. This is not effective in a real-time gaming environment that needs to be updated by searching for the best numbers right now.

The performance of most MCTS depends on how the policies of the selection and implementation phases are improved. A general approach has not been formed yet, with performance significantly reversed depending on the nature of the application environment. For the selection phase, the policy using the Upper Confidence Bound formula [15] is mainly referred to, and is currently used by default in most MCTS. The UCB plays a role in making the tree somewhat balanced, rather than biasedly expanding its navigation priorities. If you continue to expand based on the node you selected for the initial random selection and select only the best evaluation values, you will have a problem with searching based on some of the nodes found earlier in the tree navigation. To address this, a formula was proposed that would allow the scope of the search to be flexibly controlled by adding weights to nodes that have a small number of searches or that have not yet been evaluated.

The UCB formula is as follows.

$$\mathrm{UCB} = v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} \tag{1}$$

Vi is the evaluation value of the current node and np is the total number of visits by the parent node. ni is the total number of visits to the child node and C is the constant that controls the nature of the tree navigation. The higher the C value, the greater the correction for the insufficient number of searches, and the lower the C value, the less the correction for the insufficient number of searches. This is called the "Exploration and Exploitation" relationship.

The proposed plan is the Progressive History, which introduces a concept that reflects all records obtained during the tree exploration process into evaluation values. PH does not rely on knowledge-based methods because it reuses the cumulative arbitrary enforcement results for each case in a random trial without throwing them away.

$$\mathrm{PH} = v_i + C \times \sqrt{\frac{\ln n_p}{n_i}} + \frac{S_a}{n_a} \times \frac{W}{n_i - s_i + 1} \tag{2}$$

It consists of the child node evaluation values from the beginning, the middle part of the UCB, and the PH reflection part, which was finally added. Where Sa and na have different characteristics from the previous child node evaluation value Vi, not using the MCTS valuation values established through the current inversion wave, but recording and accumulating all of the evaluation values for the number of cases used in random trials until now. In other words, use the evaluation value obtained during random execution, not the evaluation value stored on the node on the MCTS that the child node has.

Weighing W values are used to determine the reflection rate of the PH portion, which is also empirically given. In order to differentiate the ratio of the W value from each node, the W value for each node is divided by the $ni- si$ value. This is the total number of visits on the i node minus the number of wins, which is the number of defeats on the i node. For additional constant 1, the constant is to avoid the situation where the denominator is zero. According to the formula, the higher the number of defeats, the lower the W value, the higher the W value, which increases the biased characteristic of the accumulated record. Currently, the higher the probability of winning the i node, the more the evaluation value is, the more the accumulated arbitrary trial value is to be reflected. This is a correction to prevent the distortion of the tree's direction of expansion, such as actively creating nodes by reflecting the cumulative enforcement value SA even if the current node's evaluation value is poor. When SA values are often accumulated where they are not related to the current node's parents, the adverse effects of selecting and expanding a node are not fatal if the existing evaluation values are good, but the adverse effects caused by past accumulated values that are not related to the current node are likely to worsen the future situation and reduce the percentage of the cumulative values.

III. STRATEGIES

In this paper, when the Progressive History technique is first applied, it seeks to obtain the interval of the initial weight value with the highest average odds, and to obtain a strategy to overcome the fixed odds in the initial weight value and find the best odds in real time through the dynamic formula that reflects it in real time. Existing MCTS algorithms also perform actions to find the best odds through the selection phase, but considering only the best winning values during the selection phase leads to problems of inefficient behavior, such as the algorithm being buried at the local maximum value. To compensate for this, it is a common approach to improve performance at the selection stage through dynamic UCB formulas that have improved PH formulas, etc., and to find global maximums more efficiently. However, the C and W parameters in the formula are empirically obtained and entered for each application environment, and the criteria are not clear. The most winning weight values from the development test set provided for the GGP may be obtained in advance, but weights are not empirically available for any test set presented at the GVGAI’s main line. In other words, the traditional approach to fixing weights presents a performance-determined problem for each test environment in which it is run.

Therefore, in this paper, using the test set given as an example, we are going to ask through the experiment what range of C and W weight values are empirically favorable for each test, and propose a way to reset the weight values favorable for each test in real time.

A. GVGAI Play Environment

The execution environment has a limited time of 40ms per unit 1 tic (or 1 frame) AI agent capable of behavioral and AI counting, even if the game is typically over but the time-out is within 1000 ticks. Based on the 2000 game, the maximum run time given is 80000ms. Most of the time, winning or losing before, and reaching 2000 ticks does not always mean losing over time. In some cases, the conditions must be maintained until the 2,000-ticks are reached after satisfying the conditions for victory. The tests presented in the competition are one-man and two-player play games, and this paper conducted experiments in a one-man play game environment. Even a single-player game is static and rarely wins, and the difficulty of the test varies enough to measure the superiority of performance. While all environments have static games, such as the Wayfinding Game, the number of cases is sufficiently complex and can be classified as non-decisive if there are multiple NPCs that move randomly independently of the player. In the former case, full navigation in place without using MCTS is likely to win, but in the latter case, calculation in place until the end of the full search will force minimum actions to detect and avoid risks each time because there is a situation where the NPC attempts to disrupt and defeat the player. Although it is difficult to predict and explore the behavior of the unpredictable NPCs every time, the behavior of the NPCs changes every time, due to the behavior of the NPCs that were calculated in advance at the previous stage, is no longer valid and likely to be wasted at the next stage.

For all 82 tests, there are five levels of difficulty, ranging from very easy to very difficult, and there are changes in the form of maps and the number of NPCs and objects. In this paper, the experiment was conducted and analyzed based on the level 1 difficulty, as the experiment determined that the probability variety was sufficient to be worth analyzing the performance of the algorithm even in phase I. MCTS policies in both the selection and implementation phases followed the same way as the basic algorithms of MaastCTS2. As a result, the values C and W were minimized to the extent that they did not affect victory or defeat, and the evaluation values reflected in the reverse wave phase were divided into +1, -1, and the scores were not affected by the score, but the scores were better.

B. Dynamic Parameter Adjustment Strategies

First, a strategy was proposed to reset the weight value periodically to cope with the problem of frequent timeouts by having the same pattern of behavior repeated throughout a given area by the weight value initially set. The main purpose of this strategy is to change the values C and W at a given interval of ti, so that they repeat the same behavior and fall outside the scope of timeouts. The aim is to avoid falling into the pattern of action by inducing to the extent of the weight that does not occur. For each adjustment, one search is performed along a total of nine weight combinations by setting the search range from the existing C values to +1, existing values, -1, existing W values, and -1 ranges. The existing weights are reset in real time by selecting the combination of the C and W values with the highest evaluation values through PH calculations. By adjusting the weight of PH, the variables of the UCB calculation formula used in the selection phase are dynamically applied, and in real time they are close to the formula in favor of victory. After analyzing the validity of the real-time weight reset strategy by applying it to real-time games, there were many games where the odds were significantly improved, but there were also many games that did not have much impact, especially for the type of games that were quickly determined in the early stages of the game.

If a reset strategy was used to respond to situations in which early wins and losses were quickly determined, the C and W values should be in the appropriate range as soon as possible, but the periodic reset alone was not sufficient. In the case of zenpuzle, on average, victory or defeat is determined when 51.41 ±20 ticks are reached, so it is necessary to reset to the effective range before about 30 ticks. However, setting the reset cycle too short is not necessarily an appropriate method, as the existing reset strategy makes it difficult to maintain the optimal cycle already found for games that can produce good results. Therefore, by attempting to reset each time at the beginning of the run only to a certain time, a weight pre-emptive strategy is proposed in which the initial weight value falls within the favorable weight range in a fast time in a win-sensitive test. In the case of zenpuzzle, it is a dependent test environment in which the odds vary markedly according to the value C, and unlike realsokoban, the odds are also very high when the value C is within range. On average, however, after 51st-ticks, no action is allowed, and they lose out over time. To resolve this, it is necessary to reset the stable C value at least before a win or loss is determined. The effects of setting the pre-emptive period tp at the start of the new game were tested in real-world games and the results of using this pre-emptive strategy looks favorable for games that did not work well with the weight reset strategy only, but in some cases the winning rate was reduced, especially for the games that produced good results by the reset strategy alone.

When using a periodic reset strategy and a preemptive strategy, we have identified that the effectiveness of the policy is contradictory, as in the case of the aforementioned zenpuzzle and realsokoban. In the case of zenpuzzle, it is necessary to actively reset the C value early on because of the low probability of winning the initial C value. However, there is a problem that contradicts the response to realsokoban, which is more advantageous if there are fewer changes in the initial C value. As a result of the review, it is expected that if the initial application of the pre-emptive strike is delayed by td and the width of the change to the current 11 Index is changed from 50% of the delayed td time to ±2 Index, the C value can be quickly reset to 2.0 or higher even if the pre-emptive strike is delayed briefly. For example, in experimental environments set to ti=20, tp=10, td=6, a pre-emption is required for 10 ticks, but six ticks are set to be delayed. Therefore, the pre-emption does not work up to the 6 ticks. However, it is required to reset the pre-emptive front-point of each tick with a variation of ±2 from 6 to 6+6/2=9 and to reset the pre-emptive front-point of the change of ±1 Index from 10 to 6+10=16 Ticks normally. After that, resetting the cycle is performed normally every 20 ticks.

IV. EXPERIMENTS

In order to measure the effectiveness of the strategy proposed in this paper, we measured the changes in the winning rate according to each case, changing the values of the ti, tp, and td. Table 1 shows the values of the variables with the best three odds for the blacksmith game. Here you can see the best winning rate is 60% when the values for the ti, tp, and td are 50, 35 and 15, respectively.

TABEL I. GAME: BLACKSMOKE

Winning Rate	ti	tp	td
60%(Best)	50	5	15
10%(Worst)	10	10	5
0%(Static)	0	0	0

Table 2 is the case for the firecaster game, in which case best winning rate is 100% while the values of ti, tp, and td are 20, 20, and 15.

TABEL II. GAME: FIRECASTER

Winning Rate	ti	tp	td
100%(Best)	20	10	15
40%(Worst)	5	35	5
20%(Static)	0	0	0

In order to measure the effectiveness of the strategy proposed in this paper, we measured the change in the winning rate according to each case, changing the values of the ti, tp, and td. Table 1 shows the values of the variables with the best three odds for the blacksmith game. Here you can see the best results when the values for the ti, tp, and td are 50, 35 and 15, respectively

Depending on the type of game, the optimal variable values vary, but we can see that the overall results are better than the proposed strategy of self-reliance. After experimenting with the effects of variable values in each game, the value of the variables with the best odds was obtained and the value of ti value 50, tp value 35 and td value 5 were obtained. Using these three strategies, the final experiment was designed and the odds measured by applying them to the entire test environment.

The final experiment also confirmed that the average winning rate for the entire 82 tests was valid and resulted in a slight increase from 57.40% to 60.84%. Calculating the odds of a valid group of experiments, except where the odds are 0% or 100% for both before and after the experiment, resulted in a further increase from the previous 48.94% to 55.82%. Accordingly, the three dynamic UCB strategies introduced for real-time adaptation of C and W values have been identified to have a positive effect on improving the uniform performance of the average odds over the entire test.

For the experiment, we use the hardware consisted of Intel i5-6600 3.30GHz (4Core 4Thread) CPU with 16G memory using two DDR4 8G PC4-19200 (2400MHz) RAM. Test software was designed and implemented with the JAVA platform provided by GVGAI, and the other execution environment followed the underlying environment provided by the platform.

V. CONCLUSION

In this paper, we tried to find a way to adopt MCTS algorithm to the dynamic, first encountered game, and propose to adjust progressive history approach by adjusting its C and W constant in the real time during the game play period. As a result, we can find a way to dynamically adjust PH parameters and optimize them to improve the winning ratio. Although some games are very sensitive to that parameter chang and some is not so sensitive, experimental results show that there is a certain improvement in the total winning rates..

When we test the proposed strategy to the total given, according to the goals of the general game play study, experimental results show that performance could be improved for some tests by introducing a periodic reset strategy, a pre-emptive strategy, and a delay pre-emptive strategy and by adjusts the values of C and W for each environment, showing that the average performance for the entire test was also improved.

Although our experimentation used heuristic initial values for the C and W, and use that value as a start point of proposed strategy, it looks possible to find the optimized values with further research and expect better performance. If environmental factors, such as knowledge-based modules, are classified and inputted to compute the range of weight values that focus on winning improvements for each input state, it is likely that the appropriate values can be estimated and adapted in the same way as in this paper. This can be seen as similar to a human player's previous experience of playing similar games.

On the other hand, further study from a different point of view is likely to be needed for tests with little win-loss change in the C and W values of existing tests. For problems where the number of valid cases in which scoring occurs exists at a very deep stage, so that tree navigation cannot be reached, or that require more effective navigation methods to be introduced during the implementation phase, improvements in the implementation phase are likely to be more effective, such as adjusting the depth of the exploration of the NST strategy, adjusting the proportion of the LA strategy, or applying the cumulative bias values to the implementation phase by adopting the RAVE strategy. In the future, through these general game play studies, we will be able to learn more about non-learning environments.

REFERENCES

[1] Chaslot, Guillaume Maurice Jean-Bernard Chaslot. Monte-carlo tree search. Diss. Maastricht University, 2010.

[2] Browne, Cameron B., et al. "A survey of monte carlo tree search methods." IEEE Transactions on Computational Intelligence and AI in games 4.1 (2012): 1-43.

[3] Świechowski, Maciej, et al. "Recent advances in general game playing." The Scientific World Journal 2015 (2015).

[4] Levine, John, et al. "General video game playing." (2013): 77-84.

[5] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[6] Perez-Liebana, Diego, et al. "The 2014 general video game playing competition." IEEE Transactions on Computational Intelligence and AI in Games 8.3 (2016): 229-243.

[7] Bontrager, Philip, et al. "Matching games and algorithms for general video game playing." Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference. 2016.

[8] Perez, Diego, Spyridon Samothrakis, and Simon Lucas. "Knowledge-based fast evolutionary MCTS for general video game playing." Computational Intelligence and Games (CIG), 2014 IEEE Conference on. IEEE, 2014.